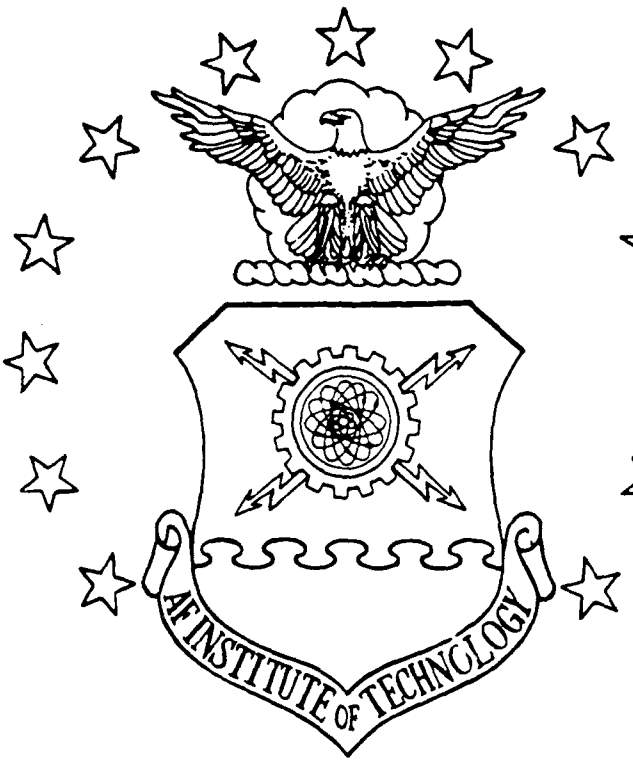

SAMPLING DATA ONTO RECTANGULAR GRIDS FOR VOLUME VISUALIZATION

THESIS

Raymond Phillips Lentz, III
First Lieutenant, USAF

AFIT/GCE/ENG/89D-4

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCE/ENG/89D-4

SAMPLING SCATTERED DATA ONTO RECTANGULAR GRIDS FOR VOLUME VISUALIZATION

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Computer Engineering

Raymond Phillips Lentz, III, B.S.
First Lieutenant, USAF

December, 1989

Preface

The purpose of this study was to develop a method for sampling three dimensional scalar data onto a rectangular grid for image display. Visually analyzing data using computer graphics techniques such as volume rendering is rapidly gaining popularity with scientists and engineers.

Many techniques for rendering volume images capitalize on the regular structure of rectangular grids. However, a rectangular grid is often not the most convenient structure for computer simulations using finite–element analysis. This is particularly true for computational, or numerical, simulations involving fluid flow.

In this thesis investigation I developed two methods for sampling scattered data and provide two filtering functions for one of the methods. Testing was accomplished using data from an actual fluid flow simulation. The results are very promising and the work warrants further investigation.

During the course of this thesis I received a great deal of help from many people. I owe thanks to Maj Phil Amburn, my faculty advisor, for his support without which I would have been lost much of the time. I also owe thanks to Mr Bob Conley and the Air Force Weapons Lab for their generous support of this research. Special thanks are in order for Dr Phil Webster and Mr Stephen Scherr from the Wright Research and Development Center for supplying the test data, and answering many questions. I also owe thanks to Dr Gary Lamont and Capt Phil Beran for their guidance and suggestions. Finally, I want to thank my wife for her support and understanding during the long months spent at AFIT.

Raymond Phillips Lentz, III

Table of Contents

List of Figures

List of Tables

Abstract

Three dimensional arrays of scalar data representing spatial volumes arise in many scientific applications. Analysis of this type of data is difficult because of the size of the data sets. Computer graphics techniques for rendering images of three dimensional data have been developed recently.

In computational fluid flow analysis, methods for constructing three dimensional numerical grids are being refined. This technique is particularly suited for simulations involving finite element analysis. The three dimensional grids produced by these methods are generally not rectangular in shape.

Many graphics methods for rendering three dimensional volume images take advantage of the physical structure of rectangular grids. Because finite element analysis is useful in fields other than fluid flow analysis and the numerical grid has promising applications, methods for handling arbitrarily shaped data grids are needed.

This thesis investigation develops a method for sampling data in virtually any form onto rectangular grids. Two sampling methods are developed and implemented using two different sampling filters. The results were successful in rendering images of both fabricated data and data from a fluid flow simulation.

Input data distribution characteristics affecting the sampling techniques were identified, and possible solutions are provided. The goal of this study was to determine if data could be successfully sampled from a grid of arbitrary shape onto a grid of rectangular shape. The results indicate that this is indeed possible.

SAMPLING DATA ONTO RECTANGULAR GRIDS FOR VOLUME VISUALIZATION

I. Introduction

1.1 Background

Three-dimensional arrays of scalar data representing spatial volumes arise in many scientific applications (5). Volume rendering is a relatively new graphics technique for visualizing this type of data. Researchers can analyze visible and invisible phenomenon present in virtually any type of data collected, or computed, in three-dimensional scalar form using volume rendering. This method of visual analysis promises to greatly reduce the time required to analyze the rapidly growing number of data sets produced by computer simulations. It also has promising applications in the medical field in areas such as computerized axial tomography (CAT) (7).

Much of the published literature focuses on visualizing medical data, but the algorithms used for volume rendering are not dependent on the source, or type, of data. Volume rendering algorithms have typically been implemented using either cell processing or voxel processing techniques(23:60). A universal definition for the two techniques has not been established, but this study accepts the definition given by (24). Cell processing techniques assume that each cell, or volume element, has functional data values associated with each of its eight vertices, which vary continuously within, or outside, the cell volume. Voxel techniques, on the other hand, assume a single functional value is associated with each volume element, and has a constant value throughout the element.

Cell processing techniques have been implemented in two ways (23:61). First, the functional values can be integrated through each cell, and their contribution to the image is found by projecting the integration onto the two–dimensional viewing screen. The second way to process a volume cell is to point sample the data using a form of ray tracing. In general, the faster of the two methods has been the simplified form of ray tracing known as ray casting (23:60).

1.2 Purpose

The purpose of this study is to design a method for sampling data onto rectangular grids for visual display using existing volume rendering software. The data may be organized on arbitrarily shaped curvilinear grids, or be scattered in space with no particular structure. The system used to generate images from the transformed grids is the Volumetric Imagery Program for Engineering Research (VIPER) developed at the Air Force Institute of Technology by Captain David Bridges (1).

1.3 Problem

Much of the emphasis in developing volume rendering algorithms has been focused on the physical structure of the data. Medical data produced by X–ray Machines or Computer Axial Tomography (CAT) scans is two–dimensional. These two–dimensional slices can be combined to form three–dimensional data sets (13). If these three–dimensional data sets are shaped as rectangular meshes, a ray casting, or other cell processing algorithms, can take advantage of the structure to speed up the grid traversal during the image generation process. Unfortunately, not all scalar data grids are structured in rectangular meshes.

A popular grid construction technique used in fluid flow research is the numerically generated grid (21). Because of the recent advancements in computers, fluid flow researchers can use these grids to simulate many problems previously requiring models and wind tunnel testing. The grids are constructed around the physical

object under test, and are transformed to use standard partial differential equation solving routines. The transformations can be either continuous or in discrete tabular form. They are applied to the equations used to calculate the data, rather than on the physical grid points. This means the final value of the calculated data is independent of the transformations, and the results are three–dimensional grids of arbitrary shape (21).

Computational fluid flow researchers produce massive quantities of scalar data structured in arbitrary shape. Much of the impetus for numerical grid generation techniques came directly from the field of computational fluid dynamics (21). Numerical grid generation is still a relatively new technique, and as advancements are made uses for these grids are likely to spread to other fields of research. For this reason, new methods allowing visualization of data arranged in non–uniform, three–dimensional meshes are needed.

1.4 Scope

VIPER uses ray casting to generate images and depends on rectangular, orthogonal, three–dimensional grids. The volume traversal algorithm takes advantage of the uniform structure of rectangular grids. A ray direction and starting point provide enough information to rapidly determine a location within the grid. VIPER does not require a data set to be defined on a rectangular grid with uniform spacing between the planes, but it does require that all the planes along each principle axis in the cartesian coordinate system be parallel to each other and orthogonal to the axis. The the number of planes is limited only by the computer memory resources available.

A method of transforming a data set into a form VIPER processes directly is developed in this study. This transformation is a separate pre–processing step, and does not affect the time required to render images. The reason for making this a separate processing step is simply to save rendering time. A data set need only be

transformed once, but many images can be generated from that data. The method developed is a technique for sampling data onto a rectangular grid.

All the software developed is written in C and operates under UNIX, but is generic enough that it can be easily transported to any Machine having a C compiler. The C programming language was chosen because it is the language used most in computer graphics. The UNIX operating system was used because of its availability and its support of the C programming language. The output format for the the rectangular grids is designed to satisfy VIPER's input requirements, but can be easily modified for use with other volume rending software packages.

1.5 Approach

This project was accomplished in several separate, but not necessarily independent steps. The VIPER program was thoroughly analyzed, a search of current literature was conducted, a requirements analysis was performed, and a system design was implemented. In addition, testing, validation, and documentation of the code was completed.

Analysis of VIPER. The VIPER program was analyzed to gain an understanding of its volume rendering technique. This was necessary for two reasons. First, modifications were necessary in order for the program to accept transformed data. Second, a complete understanding was necessary to use the program to generate images of the transformed data.

Literature Search. Current literature on ray tracing techniques and volume rendering methods was reviewed. The goal of the review was two fold. First, the background necessary for the requirements analysis was established. Secondly, an understanding of ray tracing techniques and volume rendering methods was necessary to determine the appropriate method for handling arbitrary grids. The results of the literature review are presented in Chapter 2.

Requirements Analysis. The requirements for sampling scattered data were defined. The goal of the analysis was to determine the functions and parameters necessary to implement the program. The tool used to accomplish the requirements analysis was the Air Force Materials Lab's $IDEF_0$ requirements analysis language. $IDEF_0$ is explained in Chapter 3, and the analysis is presented both in Chapter 3, and Appendix A.

System Design. A design was developed for sampling scattered data onto a rectangular grid based on the requirements analysis. The tool used to accomplish the design was the Structured Analysis and Design Technique (SADT) diagrams. The system design is presented in Chapter 4 and Appendix B.

Implementation. The design for sampling scattered data was implemented on a scientific workstation. A workstation was used because of availability, and previous familiarity with the Unix operating system. The workstation is quickly becoming the tool of choice for both scientists and engineers. This system is more flexible and useful by operating within the workstation environment. The detailed design specifics are presented in Chapter 4.

Testing and Validation. The design was tested against the requirements, and the results were validated using data from a computational fluid dynamics finite-element analysis. Because of the size of the data sets, and the processing time involved, two data sets were chosen for testing. The images were compared to results from NASA's PLOT3D (25) image rendering software. The scientists supplying the data provided coments on the images. The testing and validation results are presented in Chapter 5.

System Documentation. A user's guide is informally presented in Chapter 4 with the system detailed design. It describes not only how to use the grid transformation routine, but also changes made to VIPER. Fully documented source listings for the grid transformation package can be obtained by contacting the Electrical and Computer Engineering department at the Air Force Institute of Technology.

Results and Conclusions. Finally, the results of this project are discussed in Chapter 6. The tool developed in this project is a data transformation tool. The success of the tool is discussed from a qualitative perspective. No attempt is made to validate filtering, or implemented convolution methods. Recommendations for future research and enhancements are discussed based on the findings in this project.

II. Logical Development

2.1 Introduction

A logical development for the sampling program is presented in this chapter. Section 2.3 contains information necessary to familiarize readers with ray casting techniques. Ray tracing improvement techniques are briefly discussed in Section 2.4 to establish the background necessary to justify design decisions. Finally, a review of current literature on volume rendering methods is presented in Section 2.6.

2.2 Background

The goal of this study is to sample data from grids of arbitrary shape onto a grid of rectangular form. To define a rectangular grid, first consider a grid in two–dimensional cartesian space. The grid coordinate system is orthogonal, and obeys the right hand rule. The grid is formed with straight lines that are parallel to one coordinate axis, and perpendicular to the other. The intersection of these lines form the vertices of rectangular cells within the grid. The lines can be variably spaced along either coordinate direction, but only one line can be at a given point along the axis. The grid lies in one plane in the coordinate system. All intersecting lines form either 90, or 180 degree angles with every other line at intersection points.

A three–dimensional grid can be formed using two, or more, of the two–dimensional grids. The grids are stacked on top of each other with only one grid occupying any given plane. They are connected with straight lines at every vertex of every cell. If the three–dimensional grid satisfies the definition of the two–dimensional grid when viewed perpendicular to each of the six faces, it is a rectangular grid. Figure 5 contains a photograph of a three–dimensional grid used during this project.

Before discussing how to perform data sampling, it must be shown that grids of arbitrary shape cannot be easily handled using ray casting techniques. It is not

necessary to sample data from an arbitrarily shaped grid onto a rectangular grid to generate images of that data, but methods used to generate images can benefit from the regular structure of a rectangular grid.

2.3 Ray Casting

Ray tracing is a graphics technique which point samples data to create images. Ray casting is a form of ray tracing where rays do not spawn new rays at object intersection points. In a full ray tracing algorithm, a ray is traced from a pixel along a straight line into the volume until it intersects an object. From that point, several new rays may be generated with different directions. All the rays, and their spawned rays, have some effect on the color assigned to the originating pixel. The surface characteristics of the object also has some effect on the pixel color and the spawned rays' directions. In a ray casting algorithm rays are traced in the same manner, but the intersections with objects are different.

For opaque objects, the color contribution is found at the intersection point and the ray terminates. If an object is not opaque, the color contribution is found at the intersection point, the ray's light level is adjusted according to its transparency, and the ray continues along its original path. The VIPER program was implemented using the ray casting method for volume rendering.

VIPER assigns a light intensity value to each ray, and as the ray is traced through the volume its intensity decreases at every target intersection point based on parameters set by the user. This attenuates the rays as they travel through the volume. When the ray's intensity is small enough so that it cannot contribute a significant value, or it exits the volume, the tracing is terminated. The cumulative color from each intersection is then added and assigned to the pixel from which the ray was cast.

This technique allows transparent surfaces to be rendered. Rays are attenuated at object intersection points by a value between one and zero, where zero is

completely transparent, and one is completely opaque. Ray casting techniques are computationally faster than standard ray tracers, but cannot provide all of the same image enhancing features (23:61).

To trace a ray through a data grid, VIPER finds the starting position of the ray, and uses the ray's direction to *step* through the volume one cell at a time. At every cell intersection, the data value at the ray's entry and exit point is found by linearly interpolating the data values from each vertex in the cell wall. This allows the data values at the ray's entry and exit points to be compared with the target values. If a target value lies between the entry and exit values, the ray is processed to compute the color contribution.

Much of the CPU time spent in ray tracing is in calculating the location within a volume, and finding object intersection points. Ray tracing is a popular graphics technique for rendering images of objects modeled with a variety of geometric primitives, and much research has been done to find ways to improve the technique. The next section reviews several popular techniques for improving ray tracing algorithms.

2.4 Ray Tracing Improvement Techniques

To follow the argument presented against ray tracing an arbitrarily shaped grid, it is necessary to develop a fundamental understanding of ray tracing algorithms. The following discussion provides basic information on ray tracing techniques, and associated problems. Techniques for solving these problems are presented to contrast how they are, or are not, applicable to ray casting methods.

Several techniques have been developed to speed up ray tracing algorithms. Many techniques focus on image improvement, which is directly related to the computation time required to generate images. Ray casting is a simplified form of ray tracing, but because of differences, improvement techniques for one may not be applicable to the other. Ray tracing is more popular than ray casting, because it produces higher quality images (8:16).

Much of the CPU time spent in a ray tracing algorithm is in determining ray locations, and object intersection points (9:19). Many ideas have been developed to decrease the computation time needed for these two operations. One of the simplest techniques to reduce the time for locating objects is to use a bounding volume to limit the search space.

Bounding Volumes. The bounding object used most often is a sphere (9). A sphere is used because of the simplicity of the geometry describing it; it is easy to determine if a ray falls within the boundary of a sphere. VIPER uses the six faces of the grid volume as a bounding object. Comparisons are made with the six faces at each pixel location before a ray is cast. This allows a rapid determination of a pixel's color when the data volume's projection does not affect it.

Digital Difference Analyzers. The second computationally expensive operation is calculating a ray's position within a three–dimensional grid. These calculations generally involve floating–point computations, and can require several bus cycles to complete. Integer calculations can be accomplished in as little as a single bus cycle on many computers. If a ray's position can be found using integer arithmetic, the positional computation time can be significantly reduced.

Each grid cell must be processed in a volume rendering algorithm to find target data values. A method for quickly stepping a ray through the grid is to use a three–dimensional digital difference analyzer (3DDDA). A digital difference analyzer is a method for drawing lines on a screen display using integer calculations. One way to implement a 3DDDA would be to create two, two–dimensional analyzers working in mutually perpendicular planes (8:18). A 3DDDA calculates a ray's path using integer arithmetic in the same way a line generator does. This method is more efficient than using floating–point operations, but has disadvantages.

The 3DDDA algorithm depends on the ray's origin and end point being in the center of a volume cell. Another problem is the cells must be of constant length along each axis direction. They do not have to be cubic in shape, but they must

be the same size because of the assumptions made by a line drawing algorithm (16:24). This technique cannot be implemented in a volume rendering program without constraining the structure of the input data.

Adaptive Subdivision of Space. An alternative to reducing the time required to locate position and intersection points, is to reduce the number of samples. If the number of rays used to build an image is reduced, the time required to construct that image is also reduced. Caution is required, because of the aliasing problem due to point sampling. A widely accepted method to reduce aliasing is to sample at higher rates (13:33). Techniques for determining the position of those extra samples have been developed for ray tracers.

If the scene being sampled is not homogeneous, the time required to process each pixel varies greatly. In a non–homogeneous scene CPU time is wasted by tracing several rays in areas where the scene is not changing rapidly. However, image quality suffers by tracing too few rays in areas where the scene is changing rapidly.

An adaptive method for subdividing the screen space can be implemented to determine how finely a scene should be sampled. This technique is applicable only when more than one ray per pixel is used to sample the scene. If a small number of samples are taken at each pixel and compared, a decision can be made whether further samples are necessary based on the color values obtained. If the individual samples do not vary greatly, no further sampling is necessary. This allows image generation using the minimum number of rays per pixel to satisfy image quality requirements. A complete discussion of adaptive sampling of screen space can be found in (28:343-349).

Fujimoto, and others, (8:24-25) argue that adaptive subdivision techniques tend to be heuristic based, and often do not result in the desired effect. The computation time spent determining the sample rate can be very close to the amount of time spent sampling at a constant rate of several samples per pixel. The result of constant sampling is some aliasing reduction traded for computation time.

Other techniques, such as distributed ray tracing, also focus on aliasing reduction while limiting computation time. According to (4), the Nyquist criteria is violated in ray tracers because samples are taken at pixel locations which are uniformly distributed. The alternative is to vary the sampling rate in some non–uniform way to satisfy the Nyquist criteria.

Distributed ray tracing methods are applicable to ray tracing, because they focus on altering the sampling with the spawned rays. Full ray tracing techniques have not been used in volume rendering algorithms principally because of the computation expense (11:VIII-15). Very few of the methods for improving ray tracers are applicable to ray casting, and those that are have been implemented in VIPER. An alternative to sampling data onto a rectangular grid would be to ray trace the data in its arbitrary shape. The next section presents an argument for not attempting such a method.

2.5 Direct Ray Tracing of Data Volumes

A grid constructed in a curvilinear coordinate system could be ray traced to produce images of the data. Several problems associated with the grid characteristics would cause an unacceptable level of overhead in both computation time, and computer memory requirements.

Curvilinear Grid Structure. Data grids constructed for computational fluid flow simulations can be either numerically generated, or constructed by hand. The grids are generally constructed in a curvilinear coordinate system, and indexed by i, j, and k, or other index variables (21). The transformations between i, j, and k curvilinear coordinates to x, y, and z cartesian coordinates is one to one, and can be analytic, or discrete (19). Each i, j, and k index corresponds directly to a unique x, y, and z value in cartesian space, however, the i, j, k indices may map to an arbitrary order of cartesian values.

At some point during the image rendering process, a transformation to screen

space is necessary. Before the screen space transformation can be performed, the coordinate system transformation must occur. If the coordinate transformations are discrete, a table must be built and maintained in memory, or accessed through a disk file. If the transformations are continuous, a computation is required for each node coordinate position. It is possible to build a table for the continuous transformations, but in the worst case both the continuous and the discrete transformations could be unique for each node point (19). In addition to the problems caused by the transformations between coordinate systems, the transformation functions are often not maintained.

When the grid is constructed, transformations are applied to the equations used to calculate the data values. This results in data values being associated with their cartesian location in space and being completely independent of the transformations. The i, j, and k indices are implicitly associated with their cartesian values by their position in the data file (19). The grid data files are written using the i, j, and k indices as loop control variables. This means the transformation functions are not needed to manipulate the data, and their presence would only serve to make the data files larger.

Finally, the cells in these grids do not necessarily have planar faces (19). The walls of the cells can be surfaces of high–order functions which do not have to be continuous throughout the grid. This would cause a great deal of error in a ray tracing program that assumes planar cell walls, and uses linear interpolation. This, combined with the discussion above, makes it clear that ray tracing an arbitrarily shaped data grid directly is not desirable in terms of added overhead computation time, additional memory requirements, and error introduction. The final alternative to sampling data from arbitrary grids into rectangular grids is a cell integrating method for rendering the images.

A Cell Integrating Method For Volume Image Rendering. A cell integrating method for rendering volume images projects each cell onto the image

screen and processes every pixel affected by the projection. This technique generally produces higher quality images because it does not suffer from the same aliasing problems that point sampling methods do.

A cell integration technique is referred to as an object–space method for volume rendering. This method can be implemented without ordering the coordinate points, but the main drawback is the tremendous number of calculations required to compute an image. Using this technique, several cells, and the pixels they project onto, can be computed independently, making the algorithm a better candidate for a parallel architecture system (24:66).

In addition, current implementations of cell integration techniques also take advantage of the regularity of rectangular grids (27). A method of sampling data from arbitrary grids into a rectangular grid could be beneficial to both volume imaging techniques. The final choice for rendering images from data in curvilinear grids is to sample that data onto a rectangular grid, and use rendering software capable of handling those grids.

Mapping Irregular Grids into Regular Grids. Mapping an arbitrary grid into a rectangular grid seems straight forward, but is not problem free. A curvilinear grid generally has very dense spacing in areas of high interest, and sparse spacing in other areas. As Figure 1 shows, the range of scales in a grid can vary and cause problems in the mapping process by increasing the number of node points.

The obvious way to map an arbitrary grid into a rectangular grid would be to project all the grid points onto two mutually orthogonal planes and use those points to define the planes for the rectangular grid. This would guarantee that a node exists in the rectangular grid for every data point in the arbitrary grid. As Figure 2 shows for two dimensions, in the area where the grid spacing is very dense those planes are extended the entire length of the volume. These planes create new intersection points in the grid which must have data values assigned.

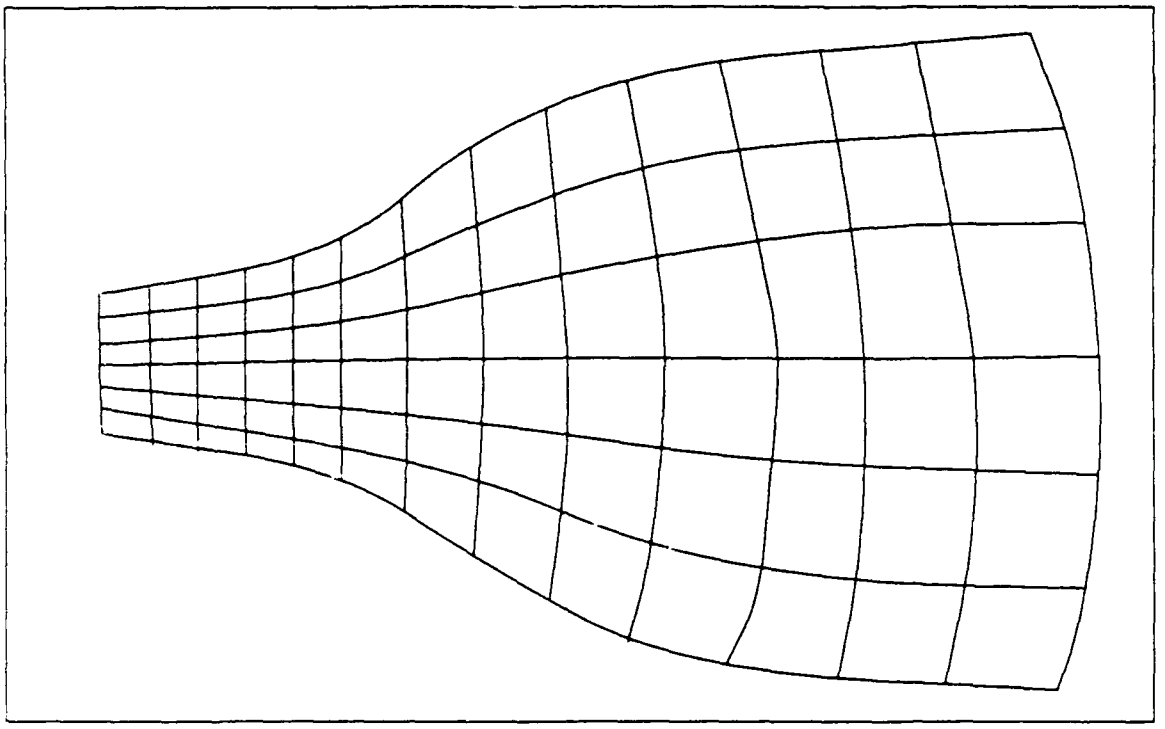

Figure 1. A Two–Dimensional Arbitrary Shaped Grid

Techniques similar to this have been implemented, but problems were encountered. Mapping a grid using this technique causes an excessive number of new nodes to be created in the rectangular grid. The extra nodes require data assignments, and that data is typically found using some form of interpolation (24:75).

In computational fluid flow research, methods of finite–element analysis are used to collect computed data. As Zoltan Cendes (3:31) points out, methods of finite–element analysis whose functions are high–order polynomials provide more accurate solutions, but are not as flexible in modeling irregular boundaries. Therefore, it is possible that for some data grids low–order polynomials are used in the simulations.

Using linear interpolation to increase the resolution of the data set should not introduce more error than is already present. Craig Upson, and others (22), have used this technique in their Application Visualization System (AVS). Marc Levoy

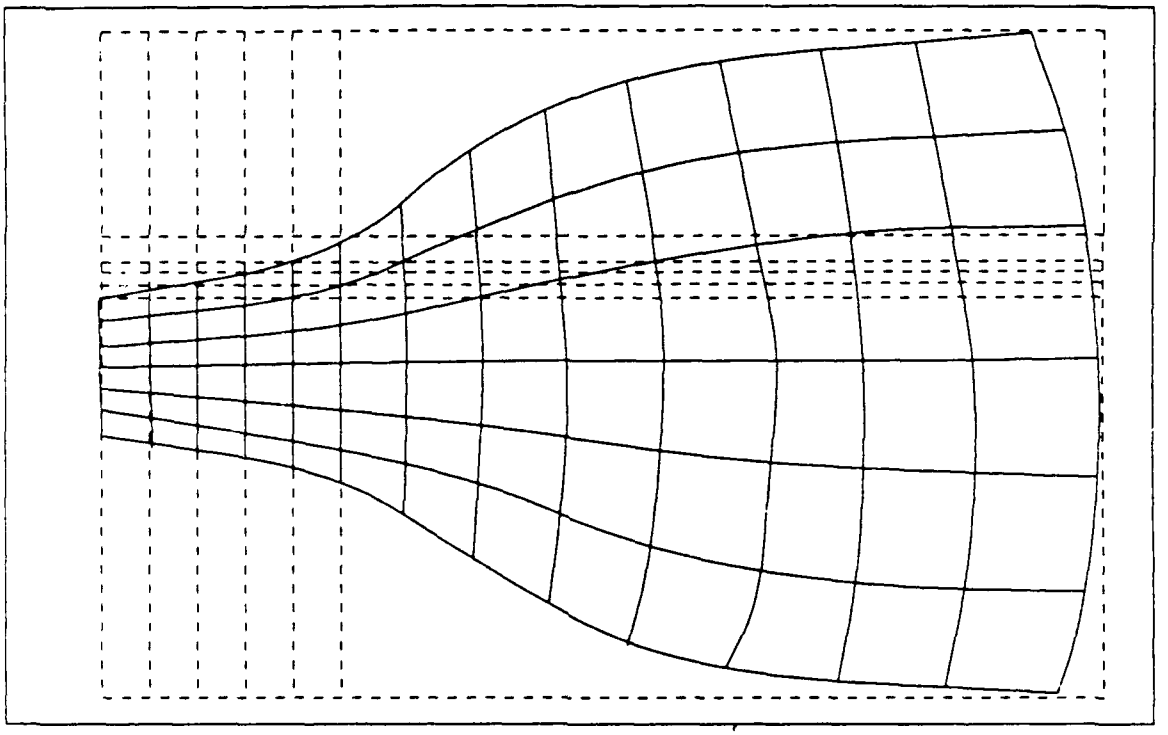

Figure 2. A Rectangular Grid Overlaid on an Arbitrary Shaped Grid

(13:34) also points out that a way to increase the resolution of a data set is to interpolate to create new data points. He says if the interpolation method is a good one, the accuracy of the visibility calculations is improved.

Once the nodes have been assigned data values, it is often necessary to filter the size of the data set (24). The grids become too large for most computer memories, and the data must be filtered to reduce the grid size. This study develops a method for sampling the existing data directly onto a previously defined rectangular grid.

Using a filter to sample the data into the grid accomplishes the same thing as interpolating, and then filtering, but in a single step. The details for this method are presented in Chapter 4. Before presenting the requirements analysis, a review of current literature is presented.

2.6 *Summary of Current Knowledge*

Early efforts at rendering images of three–dimensional, scalar data were limited by the display devices available, and were generally two–dimensional. Researchers using vector graphics display devices tried to find symbols, or other methods, to model data characteristics. As raster scan devices became available, color images were introduced, and the first three–dimensional images were rendered.

Vector Display Systems. Kenneth Kroos (12) presented a study of vector–based display visualization techniques in 1985. He found that a simple arrow–shaped vector was the optimum graphic symbol to represent vector data in a flow field. He was able to rapidly generate images of flow field data using this technique, but had several problems with data representation. First, he found it difficult to display data extremes with small display screens, and he had to generate several images to model temporal features. Finally, his system did not handle large data sets well because the resulting image was too complex for easy interpretation.

Particle tracing is a popular vector based technique of flow field imaging. This technique consists of releasing particles in a flow field and tracing them as a function of time. Particle tracing is still a very popular technique because it is relatively fast and lends itself well to user interaction. The Application Visualization System developed by Craig Upson (22:39), and others, uses a form of particle tracing called particle advection. This technique was found to be highly interactive when the number of particles was limited to no more than about 30,000. Particle tracing is also still considered a very useful technique at NASA's Ames Research Center (25).

Vector based flow field imaging systems have several advantages. These methods tend to be very fast, and provide a good environment for user interaction. They also produce sharp, crisp lines and curves, but cannot provide filled–in solid areas.

Raster Display Systems. Winkelman and Tsao (29) were among the first to report the use of raster display devices to visualize flow field data. They used

the raster device capability to represent data values with solid areas of color on the screen. They validated the accuracy of their system by comparing their results with photographs from wind tunnel tests. Although their method was successful, they were limited to displaying two–dimensional slices of the data.

Smith, Sperry, and Everton (20) extended the two–dimensional techniques of Winkelman and Tsao by generating several two–dimensional slices and combining them to produce a three–dimensional image. This technique provided additional capability over the two–dimensional method, but left gaps between the slices, forcing the user to guess about the flow field conditions in those gaps.

Raster display systems allowed for the advancement of image display devices, and led to higher resolution systems. In addition to the higher resolution systems came computer graphics techniques such as ray tracing, which could create images that exploited the ability of those systems. Better display systems, and new graphics techniques led to new ideas for rendering three–dimensional images.

Current Volumetric Rendering Techniques. A new approach to visualizing three–dimensional data was presented by William Loresen and Harvey Cline (14). They treated the data being rendered as a volume formed from regular–sized cubes, or voxels, with every voxel having a data value associated at each of its vertices. Although this technique uses a data gradient in the rendering process, it still reverts to using polygonal geometry to produce the surfaces of interest.

Their system enhanced the capability to rapidly produce images from different viewpoints. This is possible because once the polygons are fitted, simple viewpoint transformations are made before rendering addition images. Many problems can be encountered during the polygon fitting process. Methods independent of geometry were also developed.

Marc Levoy (13) developed a volumetric rendering technique to visualize three–dimensional images using scalar data. His method does not use polygonal geometry

to model surfaces, instead it uses ray casting to directly create the images. Rays are fired through the volume, and at the intersection point of a ray and a cell, the data values at each vertex are trilinearly interpolated to find the local gradient. The value of this gradient is used to determine the color and translucence of the cell, and is factored into the final color of the pixel. The main advantages of this technique are that there are no gaps in the data and the resolution of the final image is dependent only on the resolution of the display device. Ray casting techniques are point sampling algorithms, and suffer from aliasing problems.

The ray casting method of generating images from a three–dimensional data set has been classified as a backward mapping algorithm by Lee Westover (27). He has investigated a new approach of volume rendering he calls a "forward mapping" algorithm. In this alternative approach, the data is directly mapped onto the image plane. Examples of forward mapping algorithms include the Z–buffer and the painter's algorithm. Westover's approach focuses on providing an interactive rendering environment by producing images in varying degrees of resolution.

Methods for visualizing three–dimensional scalar data tend to be designed for special applications, and are generally complicated. As Craig Upson, and others, have noted:

> Traditionally, scientists and engineers have exhibited little interest in developing these tools themselves, because they require too large an investment to learn how to use or because they are outside the scientists' area of interest. (22:3)

At Stellar Computer, the Application Visualization System (AVS) was developed. The AVS is an interactive visualization system based on an object oriented approach that should be generic enough to span most of the needs for scientific visualization. The system incorporates nearly all of the current techniques for rendering images from three–dimensional scalar data. Filtering and mapping tools are provided with AVS to transform data sets into forms suitable for the rendering algorithms.

The filtering transformations include interpolation, scaling, and warping capabilities, while the mapping transformations consist of geometric mapping for contour and surface generators. These transformation modules developed for AVS further support the idea of pre–processing the data in order to prepare it for the particular rendering algorithm.

AVS is the first attempt at a comprehensive commercial product for data visualization. Because the package is a commercial product, very little specific information on the algorithms implemented in the system is available. However, nothing found in the current literature suggests that sampling data onto a previously defined grid has been tried. In the next chapter the requirements analysis is developed for the grid mapping routine.

III. Requirements Analysis

3.1 Introduction

The system analysis and requirements analysis for a grid mapping algorithm are presented in this chapter. The specification method used to perform the requirements analysis is presented in Section 3.2. The functional requirements for the mapping program are based upon this analysis. All requirements are classified as user grid mapping requirements. They are first developed informally, then specified by enumeration.

3.2 Specification Method

The method used to document the requirements analysis was the Air Force Material Lab's $IDEF_0$ requirements analysis language. $IDEF_0$ is modeled after SofTech's Structured Analysis and Design Technique (SADT), and is used by the Air Force and other DoD agencies (10). The complete $IDEF_0$ analysis documents are included in Appendix A.

Structured analysis is designed to hierarchically decompose complex programming tasks into smaller pieces that can be easily handled. The decomposition can be based on data or processes. The $IDEF_0$ method is based on processes, and is represented by a series of functional diagrams with an associated facing page text. Each diagram represents one level of decomposition, and the facing page text provides information that cannot be easily derived from the diagram.

The top level $IDEF_0$ diagram for the grid mapping algorithm is shown in Figure 3. The diagram illustrates the concepts of the $IDEF_0$ analysis language, and provides a high level abstract for the mapping program. Data dictionaries can also be used with the $IDEF_0$ language to further define data structures and processes. Data dictionaries were not developed for this project.

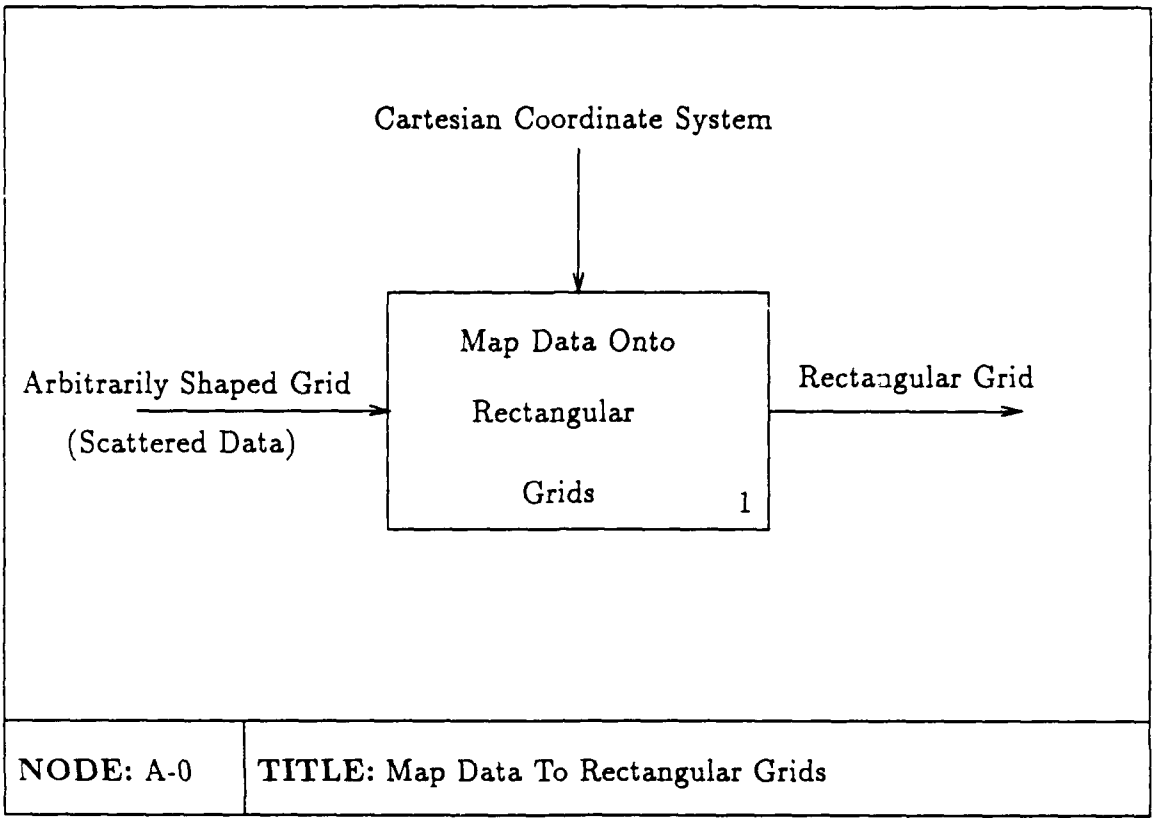

Figure 3. Top–Level $IDEF_0$ Diagram for the Grid Mapping Program

Each $IDEF_0$ diagram represents functions with boxes, and the interfaces between functions are shown with arrows. Four types of arrows can be used; input, output, control, and mechanism. Control and output arrows are required for each function, but the others are optional. Input arrows enter the left side of each box with the arrowhead touching the box. Output arrows exit the right side of the boxes, and control arrows enter the top. Mechanism arrows enter the boxes from the bottom.

These arrows show the flow of data to and from each function. Every function is considered to transform its inputs into outputs according to the restrictions of its control arrow. Mechanism arrows depict system calls the function uses to ac-

complish its task. Each function is assigned a number to help trace it through the decomposition.

3.3 Requirements Development

Users with arbitrarily shaped three–dimensional data grids have the need for a grid mapping program to take advantage of visualization software which depends on rectangular data grids. Aeronautical engineers and fluid flow researchers often perform computer simulations involving curvilinear grids, and are limited to visually analyzing their data with systems designed to handle curvilinear grids. These users also have special requirements because of the characteristics of their data grids. In general, these arbitrarily shaped grids can have a very broad range of data densities. The grids tend to be very dense in areas of high concern, and less dense elsewhere.

Informal Development. The method for constructing the rectangular grids should be optional. Two straight forward methods would be to simply space the planes in each axis direction in a constant manner between the grid extremes, or place planes where the data points are concentrated. This flexibility offers users the ability to make a trade off between computation time, and sampling accuracy.

Users must be able to select specific areas of the grid to zoom–in on for close inspection. To perform the zoom, users must be able to carve out sections of the data grid. This carving could occur before or after the grid has been mapped into rectangular form. If the user can carve the grid before the mapping process, more flexibility is provided because all the computer memory available for grid manipulation can be used to sample the carved section at a higher rate. This technique of zooming in on the dense areas of the data grids has been identified as a requirement for visualization software by both fluid flow researchers and software developers (25:IV-16). Once a grid, or a portion of a grid, is specified for visualization, a filtering function must be specified.

In order to find the "best" filtering method for a particular data set, users

should be able to select from more than one filter type. The filter parameters should also be adjustable. This flexibility allows experimentation with data sets to find the filtering function best suited for that data.

The preceding informal development of the requirements for a grid mapping system is further refined in the next section. To specify the requirements, an enumerated requirements specification is developed.

Enumerated Requirements Specification. The requirements for the mapping program should be explicitly specified before a design can be developed. The previous section presented an informal discussion of the users' requirements. Those requirements are explicitly defined through enumeration. Table 1 presents the specific enumerated requirements.

These requirements were developed based on the review of current literature, and discussions with computational fluid flow researchers. The documented requirements analysis is located in Appendix A. The detailed design implementing each of these requirements is developed in the next chapter.

Table 1. Enumerated Requirements for a Grid Mapping Program

1.0 Develop a method of mapping arbitrary grids into rectangular grids.

- 1.1 All grids of arbitrary shape, or scattered data points, must be mapped into rectangular grids.
- 1.2 Users must be able to carve sections of the grid for mapping.
  - 1.2.1 Bounding i, j, and k values for carved sections can be specified by the user.
  - 1.2.2 Carved sections of grids must be treated the same as complete data grids.
- 1.3 The method of grid construction must be optional.
  - 1.3.1 Planes can be located in areas of high data concentration.
  - 1.3.2 Planes can be located at constant intervals along each axis direction.
- 1.4 A means to filter the data onto rectangular grids must be provided.
  - 1.4.1 More than one filtering method must be provided.
    - 1.4.1.1 A density weighting filter based on distance must be provided.
    - 1.4.1.2 A Gaussian filter must be provided.
    - 1.4.1.3 A Z-buffer type of filter must be provided.
  - 1.4.2 Filter parameters must be adjustable if possible.
    - 1.4.2.1 The user must be able to specify a multiplier for the search filter radius.
    - 1.4.2.2 The user must be able to specify the number of standard deviations the filter radius represents for the Gaussian filter.

IV. System Design

4.1 Introduction

The design implementation for the grid mapping program is presented in this chapter. The detailed design was implemented using structure charts to functionally decompose each hierarchical level of the requirements developed in Chapter 3. Figure 4 illustrates the highest level of decomposition for the system.

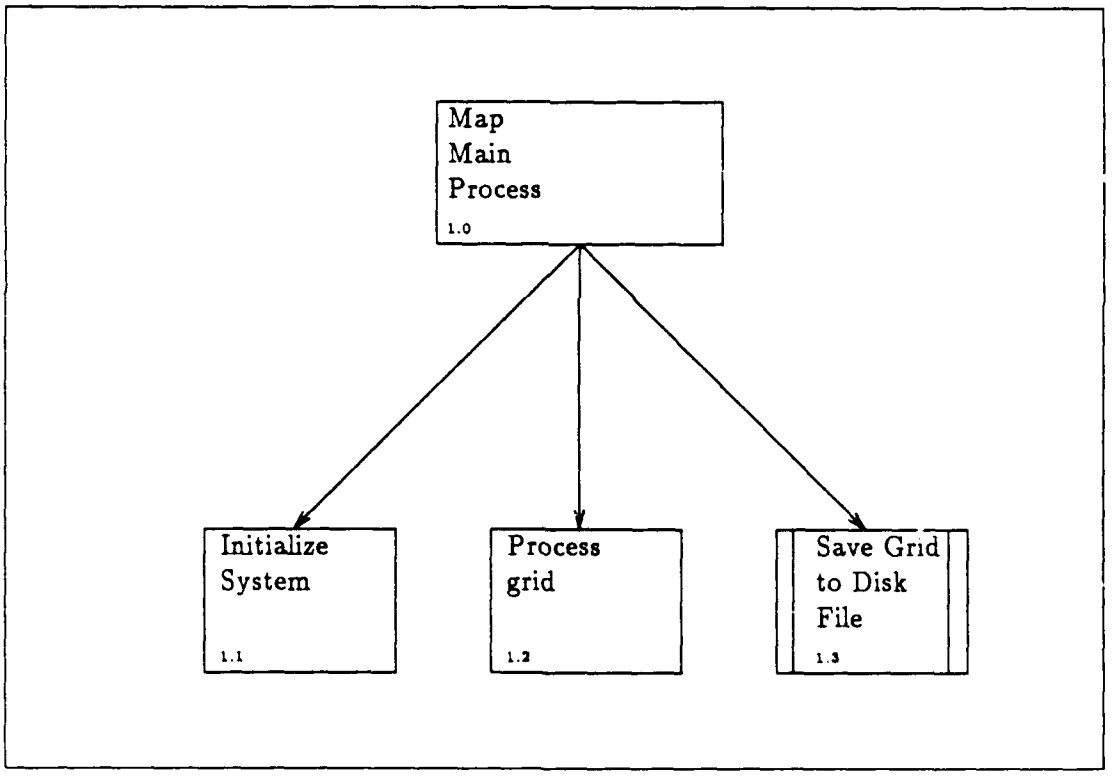

Figure 4. The High Level Structure Chart for the Grid Mapping Program

A functional decomposition was chosen for two reasons. First, the sampling process lent itself to a functional decomposition because it consists of several independent steps; examine the input data, build a grid, sample the data, and save the

output grid. Secondly, a variety of different methods can be used to build grids and sample data. By decomposing these functions into independent modules, several modules for each function can be included in the system to offer a choice between different methods. Designed in this way, the system is easily modified and maintained.

The charts diagram each functional unit's relationship with other modules in the system in two ways. First, the modules at the top of each diagram perform the supervisory, or controlling, function for each of the modules below them. Second, the diagrams show the flow of data between the different modules. At each level in the decomposition, the boxes in the structure charts assume the supervisor's number followed by a number indicating its position relative to every other box at its level. This numbering scheme makes it easy to determine the modules' relationships with each other. The complete set of structure charts, and a more detailed explanation of their function, is located in Appendix B.

The design was decomposed into the smallest possible set of related activities. This resulted in six logical sections: system initialization, carving the input grid, building the rectangular grid, the sampling methods, saving the output to a disk file, and finally, the modifications to the rendering software. Each of these design parts are discussed in the following sections. The last section in this chapter provides instructions for compiling and running the program.

4.2 System Initialization

The system is designed to give users flexibility by allowing the manipulation of several parameters, and requires a parameter file name as a command line argument. Table 2 illustrates the file structure, and required order of the user options. Each parameter item is numbered, and includes a brief explanation of its function. Comments are allowed in the file, and are recognized by an asterisk being the first non 'white space' character on a line. The mapping program reads the parameters

by scanning an entire line at a time into a character string. If the string is not a comment line, the particular parameter is scanned from the string using the format required by that parameter. Comments can also follow parameters on the same line, because everything after a parameter is ignored by the program.

Each parameter value is set in the file *map_ini.c*. The variables are declared globally to prevent passing values through several different modules for initialization. Error checking is difficult, because it cannot be known in advance what parameter values are reasonable without some knowledge of the input data. Some checking is provided for obvious unreasonable parameter values. The responsibility for accuracy lies with the user.

The parameters do not have to appear in the first column of a line, but at least one white space character, a tab or a space, must be present after a parameter and before anything else on the line. The file names may also contain paths to different locations on the user's disk, and will work as long as the operating system rules are followed.

Once the parameter file has been built, the program is ready to sample data onto a rectangular grid. Before the sampling occurs, the rectangular grid must be constructed based on the input data, and other parameters set by the user.

Table 2. Parameter File Structure

1. The grid index file name. Paths are acceptable.
2. The grid data file name. Paths are acceptable.
3. Output disk file name. Paths are acceptable.
4. An integer '1' or '0' for carving the grid. A '1' means carve it, and a '0' means do not carve it. If a '1' is given, the next 6 lines must an integer value indicating the i, j, and k lower and upper bounds in that order.
5. An integer '1' or '0' for which data value to view. A '1' means view pressure values, and a '0' means view the Mach number.
6. An integer '1' or '0' for the number of planes to use in the rectangular grid. A '1' means use the default value (100 planes per axis). A '0' means other values are given. If a '0' is given, the following line must contain 3 integers for the number of planes to use in the x, y, and z axis.
7. An integer '1' or '0' for the method of grid construction. A '1' means use regular spacing along each axis. A '0' means use variable spacing.
8. An integer '1' or '0' for the sampling method to use. A '1' means use the Z-buffer method, and a '0' means use the A-buffer method.
9. A floating point number for the filter range multiplier.
10. If a '0' was specified for the sampling method, selecting the A-buffer method, a '1' or '0' must must appear next to select either the density filter, or the Gaussian filter. A '1' selects the Gaussian filter, and a '0' selects the density filter.
11. If the Gaussian filter is selected, a floating point number is required specifying how many standard deviations the filter radius represents.

4.3 Carving Grids

Because of the wide range of scales in curvilinear, or other non–rectangular grids, the ability to carve specific sections is provided. By specifying the boundaries for a carved section causes the program to ignore all data and node indices outside that section. This means a particular section can be examined using all the computer memory resources available to sample that data.

After the grid carving parameters have been established, the indices must be examined to determine the range of the grid in each coordinate direction. Once this information is obtained, the program builds the rectangular grid.

4.4 Building A Rectangular Grid

Scattered data, or data on an arbitrarily shaped grid, can be located in varying degrees of density in the space it occupies. For this reason, two methods of building the rectangular grid are provided.

The first method is to place all the planes in each coordinate direction at constant intervals along the axis. The second method is to place the planes in areas where the data is concentrated. The effectiveness of one method versus the other is discussed in Chapter 5. The next two sections describe how each method is implemented.

Constant Plane Spacing. First, the minimum and maximum coordinate value for each axis is found by examining the coordinates in the grid index file. Once these values are known, their difference is divided by one less than the number of planes allocated for that axis to obtain a 'step' value. The first plane is located at the coordinate minimum, and each plane thereafter is located 'step' units from the previous one. Figure 5 is a photograph of an actual grid constructed from an arbitrarily shaped data grid using constant plane spacing.

Variable Plane Spacing. One way to place planes along each axis with

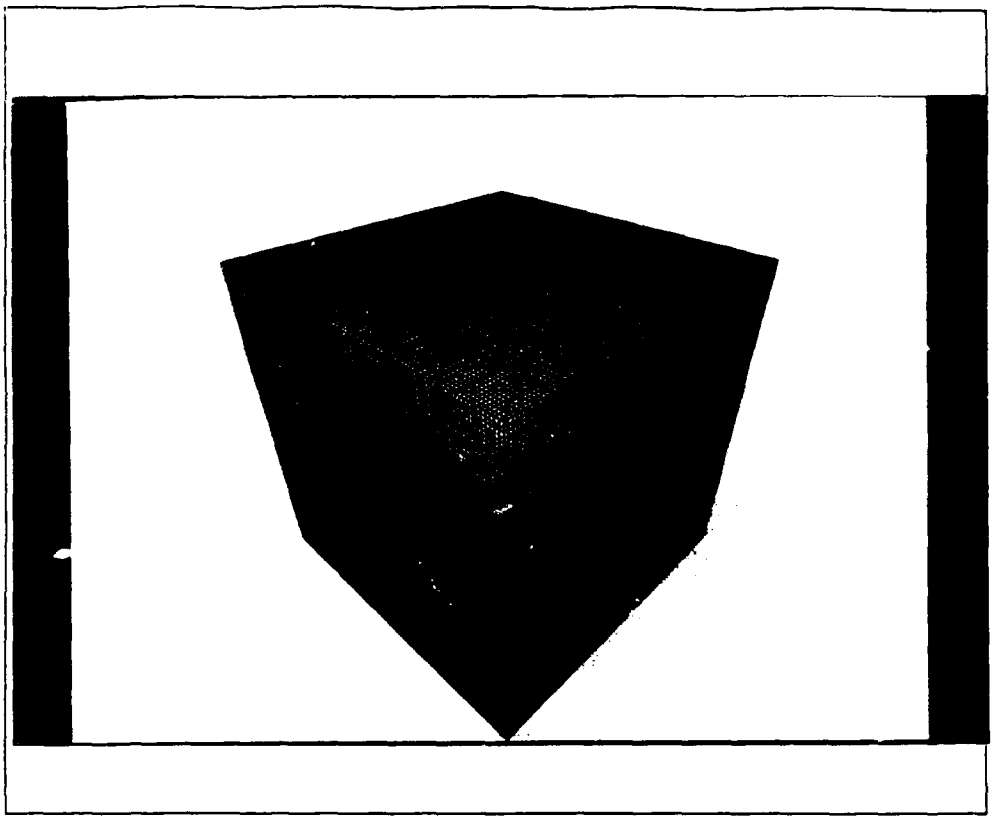

Figure 5. A Grid with Constant Plane Spacing

variable spacing requires more computation time, and memory space. First, the coordinate extremes are found in the same manner as for the constant plane method. Next, the extremes are used to construct a series of coordinate bins which contain distance ranges at each location. Figure 6 shows the structure of the bins.

The number of bins is determined by half the number of planes to allocate along each axis. The difference between the extremes is divided by the number of bins to determine the bin 'step' value. The first bin is assigned the coordinate minimum plus the bin step. Each bin thereafter is assigned the previous bin's value plus the step, making the last bin equal to the coordinate maximum.

After the bins are constructed, each coordinate value is examined, and placed in the proper bin location. The criteria used to place a value is that it must be less

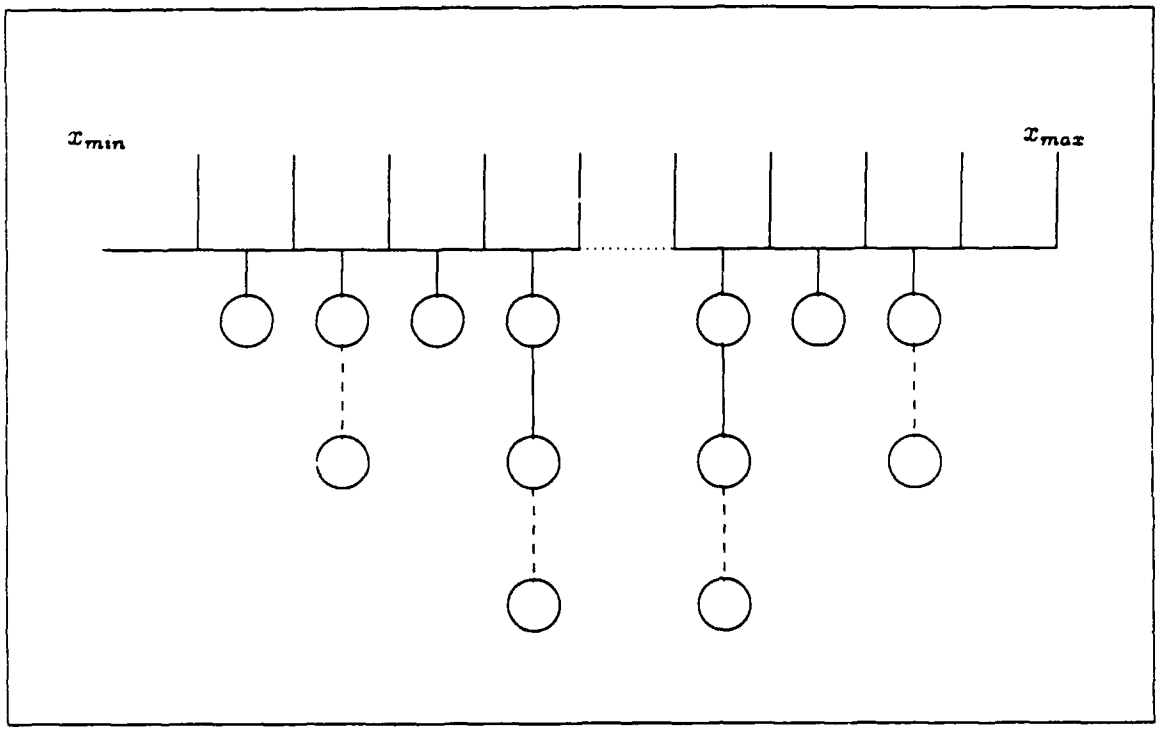

Figure 6. Coordinate Bins

than the bin value, but greater than the previous bin value. Once the correct bin is found, the value is linked into a list of the other values at that bin location. Only the unique coordinate values are kept at each bin location. Keeping only the unique values prevents allocating an excessive number of planes in a bin range if many of the coordinates in that bin range lie in the same plane.

When all the coordinates have been placed into the proper bins, the number of values in each bin is counted, and summed, to obtain the total number of unique values. The value count at each bin is maintained to determine the number of coordinates falling within the bin's range. Finally, the bin list is traversed to determine the number of planes to place in each bin range.

For every bin with a point count greater than zero, the number of planes placed in that range is found by the following equation:

$$np = \frac{bp}{tp} pl + 0.5 \tag{1}$$

where np is the number of planes, bp is the number of points collected in that bin, tp is the total number of unique points for all the bins, and pl is the total number of planes to allocate for that coordinate axis.

This method can result in one more, or one less, plane than specified, because the number of planes must be converted to an integer. To help relieve this problem, np is rounded up.

The variable plane spacing method can result in a decrease of the grid range if not enough points fall in the first and last bin locations. For this reason, when using this method, the user can force the range to be maintained. This option results in a plane being located at the coordinate minimum, and maximum, independent of the number of points in those bins. Once the number of planes has been calculated for each bin location, the grid is built.

The bin list is traversed once more, and for every bin with a plane count greater than zero, planes are positioned in the same manner as with the constant spaced planes. The plane starting position is the value of the previous bin location, and stepped through the bin range up to, but not including, the current bin value. Figure 7 is a photograph of a grid built from a data set using the variable plane spacing method.

When construction of the grid is complete, the program samples the data onto the grid using the sampling function specified in the parameter file. One of two sampling methods can be selected by setting the appropriate parameters.

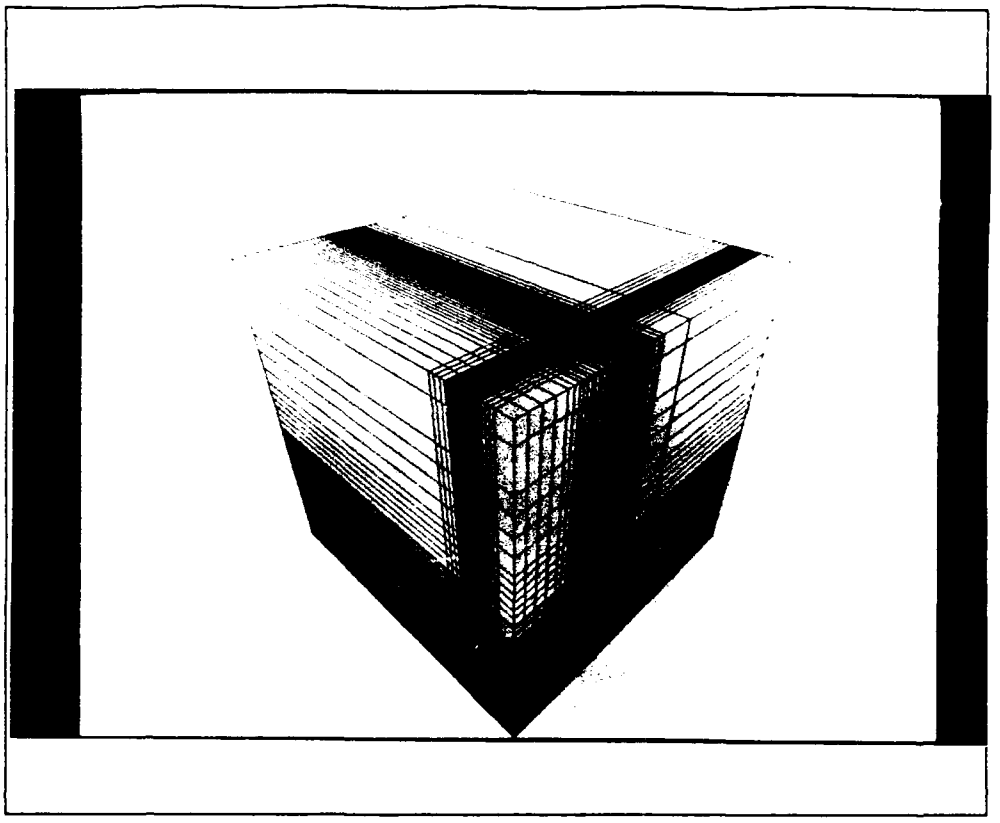

Figure 7. A Grid with Variable Plane Spacing

4.5 Sampling Methods

It is not possible to build a rectangular grid that fits the data well enough so that each node in the grid is coincident with a data point in the input data set. In addition, every node in the rectangular grid that can be assigned a data value must have one assigned. This is necessary because of the requirements of the rendering software. See Section 4.7 for an explanation of the data input requirements.

To assign data values to the nodes in the rectangular grid requires some method for sampling the input data. This sampling is accomplished using a filter to calculate data values. Two sampling methods are implemented, with one method offering a choice between two filtering functions. The first method is similar to an A-buffer (2:103-108), and the second to a Z-buffer (6:560).

An A-buffer Method. The A-buffer method for sampling data is a parametric form of a convolution filter (2). Pixel colors are determined by weighting the color of each object overlapping the pixel, based on the amount of area it occupies. The mapping program performs a similar function by assigning data to a grid node based on its distance from the true location of the data point.

For each data point in the sample data, every node in the rectangular grid falling within the radius of the filtering function receives some contribution from that data value. The filter radius is calculated using the average of the smallest and largest cell dimension in the rectangular grid. The user can specify a multiplier for the radius to change its length to suit the particular data set being sampled. By specifying a multiplier of less than 1.0, the radius length can be decreased, and multipliers greater than 1.0 increase the length

The contribution is determined from a normalized weighting term. The weighting term is calculated by subtracting the node's distance from the radius, and dividing the result by the radius. This results in weights of one for node points coincident with a data point, and zero for nodes at, or beyond, the filter radius. The weight is then multiplied by the sample data value, and both the result and the weight are summed into the node to collect the contributions from all data points at that node.

After all data points have been processed, the summed contribution term for each node is divided by the sum of the weighting terms at that node to determine the final data value. The mapping program offers users a choice between two methods for determining the weighting value.

A Density Weighting Filter. The first filtering function determines the weight value based on the square of the distance error. This value is calculated by:

$$w = \left[\frac{r - d}{r}\right]^2 \tag{2}$$

where w is the weight, r is the filter radius, and d is the Euclidean distance from the

node to the data point. The weight is squared to make the filter parabolic, rather than triangular. Figure 8 illustrates the shape of a parabolic, and a triangular sampling filter.

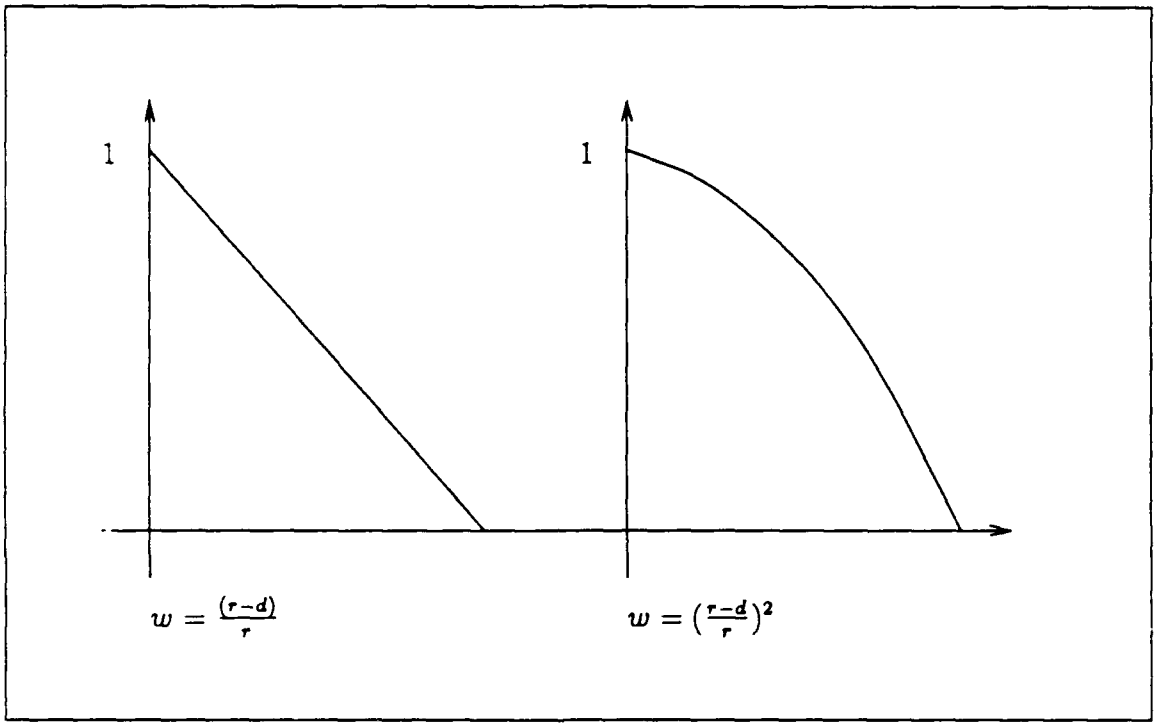

Figure 8. Density Weighting Functions

The weighting term is normalized to the filter radius to guarantee values between zero and one, and also to avoid arithmetic overflow and underflow problems.

A Gaussian Weighting Filter. The second filtering function is a Gaussian method for determining the data weights. The weights are determined by:

$$w = e^{-\frac{1}{2\sigma^2}d^2} \tag{3}$$

where d is again the Euclidean distance from the node to the data point. The term,

$2\sigma^2$, is determined from the filter radius. The user specifies how many standard deviations the filter radius represents, then $2\sigma^2$ is found by:

$$2\sigma^2 = 2\left[\frac{rm}{n}\right]^2 \tag{4}$$

where r is the filter radius, m is the radius multiplier, and n is the number of standard deviations the radius represents. Figure 9 illustrates the effect on the filtering function by varying the number of standard deviations spanning the radius.

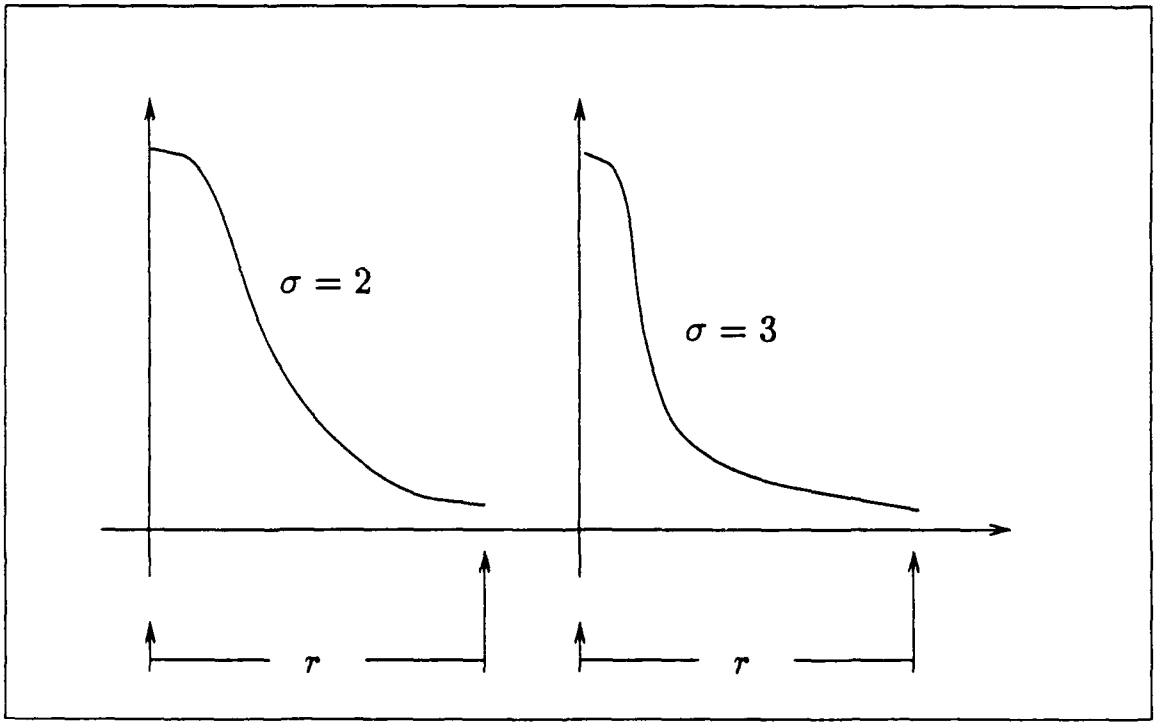

Figure 9. A Gaussian Weighting Function

A Z-buffer Method. A Z-buffer is a graphics technique for solving the hidden surface problem in image rendering (6:560). A pixel color is determined from

the object which covers it that has the smallest depth distance error. The mapping program implements a similar technique for placing data values.

When a data point is located, the grid cell that contains that data point is found. The distance from the data point to each cell node is calculated. Every node within that cell with a distance less than, or equal to, the filter radius is assigned that data value if no previous assignment has been made, or if the distance from the present data point is less than the distance from the previous assignment. If an assignment is made, the distance error is maintained at the node for future comparisons.

Once the data has been sampled onto the rectangular grid, the grid is saved to a disk file. The file name is specified by the user in the parameter file. The format of the disk file is structured to comply with VIPER's input data requirements.

4.6 Saving the Grid to a Disk File

Before saving the grid to a disk file, it must be traversed to determine which nodes do not contain data values. This is necessary because of the rendering algorithm used. The rendering algorithm assumes each node in every cell contains a continuous functional data value and interpolates from these values to determine data values at positions inside the cell.

An arbitrarily shaped grid or scattered data necessarily does not completely fill a rectangular grid built around it. This is due simply to the geometry of the grids, and results in void areas within the rectangular grid. One solution is to insert a flag value at those grid points. However, even with prior knowledge of the data values, a flag which would not insert false indications in the image can be difficult to find.

VIPER tri-linearly interpolates the data values at the point where a ray enters a cell, and at the point it exits the cell. Target values are then compared to the entry and exit values to determine if they lie within the cell. If a flag value, either positive

or negative, is located at one or more of the cell's vertices, the interpolation may result in a range which could falsely contain a target value. For this reason a different approach was taken.

Depending on the sampling method used, the largest data value is found during, or after data placement. If the A-buffering method is used, the final data values are not known until all data points have been processed. If the Z-buffering method is used, the largest data value is found during data placement. Each node in the rectangular grid contains a flag which is set during the data placement process. When data placement is complete, every node with an unset flag is assigned a value equal to twice the largest data value found.

This value is also written in the data file, and read by VIPER. During the rendering process, if VIPER detects the flag value at any of the eight vertices of a cell, that cell is ignored, and the rendering process continues. This technique prevents the use of an extrapolation method to force every node to have a data assignment. It also prevents having to build a rectangular grid contained wholly within the arbitrary grid, and losing some information outside the grid.

For convenience, the grid is translated to the origin of the cartesian coordinate system. The grid is also scaled so that at any viewing angle it fits completely within a screen of size 1024 $\times$ 768 pixel resolution. The scaling prevents users from rendering images too large and having them clipped by the screen. The next section presents changes to VIPER's file format which affect its operation on both mapped, and unmapped data grids.

4.7 Modifications to VIPER

The file structure for VIPER was slightly modified from that described in (1). The old format required the x, y, z, coordinates, and data values be listed for each node in the grid. This method resulted in file sizes exceeding 42 megaBytes for a grid of size 100 cells on a side. The format was changed to list all the x values, the

y values, the z values, and finally the data values separately. The ordering of the data values is unchanged, however, the file sizes were reduced to approximately 10 megaBytes each.

Finally, up to two additional entries are now required in the grid file. First, after the grid sizes are written, an integer '1' or '0' must be written on a separate line to tell VIPER if the input is a mapped grid. If the integer is set to '1', a mapped grid, the data flag value must appear on the next line.

These modifications were implemented on a copy of VIPER so it could be used to render images from grids generated by the mapping program. The last section in this chapter gives instructions for installing, compiling, and running the grid mapping program.

4.8 Compiling and Running the Program

The system was designed for use under the Unix environment, however, the code is written in generic C, and should compile under any system supported by a C compiler. During the testing process, every module in the program was compiled on an IBM compatible AT clone using MicroSoft C version 5.1. The program requires between 21 and 41 megaBytes of memory to execute using an input grid of size 80 $\times$ 80 $\times$ 80, depending on the type of Machine being used. On workstations with 32 megaBytes of main memory, the program executes with no memory problems, providing sufficient swap space is available.

A Unix makefile is supplied with the package to facilitate compilation under the Unix environment. Before using the makefile, issue the command *make depend* to Unix. This inserts all the local dependencies for every file in the package into the makefile. This is very useful when modifying the code, because Unix automatically recompiles only those modules changed, and every other module affected by those changes.

After the *make depend* command has been issued, the command *make* will compile the system and name the executable file *map*. To execute the program, simply type *map [-f] configuration_file*. The *-f* option will force the program to maintain the input grid ranges when using the variable plane spacing method for grid construction. It has no affect when using the constant spaced plane spacing method. For information concerning the format of the input grid, see the file *README.1ST* included in the package.

The next chapter presents the testing used for verification, and validation of the system. The validation results are presented along with color photographs of images rendered from mapped grids.

V. Testing and Validation

5.1 Introduction

This chapter contains the results of the testing and validation phase of the project. Software testing was accomplished in two steps: functionality testing, and validation testing. Section 5.2 outlines the functional testing procedures. Once the program was thoroughly tested for logical design, validation testing began. Data sets of known structure were fabricated, and images of that data were rendered. Data sets from a computational fluid flow simulation were obtained from the Wright Research and Development Center at Wright-Patterson Air Force Base, and images of that data were rendered. The images were then presented to the donators for comment. The results of the validation testing phase are outlined in Section 5.3.

The analysis of the program performance is discussed from a qualitative perspective. No attempt is made to quantify the sampling methods, or the filtering functions. The subjects of sampling and filtering are complex enough to warrant a separate study, and are beyond the scope of this project. The results using the different sampling methods, different filtering functions, and the different grid construction techniques are discussed in the last section of this chapter.

5.2 Software Functional Testing

The testing method used was one similar to that described in (18:509). Each of the lower level modules were implemented independent of each other. Driver programs were written to test the function of each module. As more modules were implemented, additional driver programs were written to test the function of the new modules. During the course of the program development, a driver program was written, and modified in a stepwise fashion, to test the function of the modules as they were integrated into a single program.

As pointed out in (18), the testing method used often depends on the characteristics of the program being developed. These characteristics may require a mix of several methodologies to satisfy the testing requirements. The mapping program was tested using criteria from more than one testing methodology. Some criteria, such as a test plan, were not necessary for this project, because its size and complexity did not justify it.

5.3 Validation Testing

According to (18:514), software validation is achieved through a series of tests that demonstrate the program's conformance with the user requirements. Validation testing for the mapping program was based on the requirements established in Chapter 3, but in addition, the validation testing considered the concept behind the program design.

From the start of this project it was not known whether sampling data onto rectangular grids would result in images truly representative of the original data. Careful research of methods previously used to visualize data from curvilinear grids provided insight upon which the assumption was made that this technique should work.

Each of the requirements outlined in Chapter 3 were implemented in the mapping program. Testing of the program based on these requirements revealed it was successful. In addition to testing against the requirements, and more importantly, the program was tested against the concept of sampling data onto a rectangular grid. This testing was accomplished in two steps.

First, data sets of known structure and content were fabricated, and images of that data were rendered. Next, actual data sets built on curvilinear grids were obtained, and images of that data were rendered.

5.4 *Validation of the Sampling Concept*

The first step in validating the concept behind sampling data onto a rectangular grid involved rendering images from known data sets. For this stage in the validation testing process, three data sets of simple geometric shapes were fabricated. The three shapes chosen were a sphere, a cylinder, and a cone. It was determined these shapes provided enough variety in structure to validate the sampling process.

Visual Validation Using Fabricated Data. Figure 10 contains photos of images rendered from a data set containing a series of concentric spheres. The data set was fabricated on a grid built in a spherical coordinate system where:

$$x = r\cos(\theta)\sin(\phi) \tag{5}$$

$$y = r\sin(\theta)\sin(\phi) \tag{6}$$

$$z = r\cos(\phi) \tag{7}$$

and :

$$0 \leq \theta \leq 2\pi \tag{8}$$

$$0 \leq \phi \leq \pi \tag{9}$$

The radius, r, was increased by units of one starting at one to obtain a series of concentric spheres. The value of the data items placed at each grid point was not important during this phase of the testing. The important point was that the structure of the input data grid was retained during the sampling process. Each sphere was assigned a different data value to allow several spheres to be viewed at once.

As Figure 10 shows, the structure of the spheres is maintained during the sampling process. A few of the data values were chosen to illustrate the structure of the spheres lying one inside the other. The outer spheres were rendered partially transparent, and the inner most sphere was rendered completely opaque. The lines in the image which appear to form edges of rectangular cells are due to the rendering

algorithm used by VIPER. Those lines interfere with each other and cause the illusion of circular patterns. This problem is discussed in the last section of this chapter and in the next chapter as well.

To further illustrate the physical location of the spheres with respect to one another, an image with a portion carved away was rendered. The lower photo in Figure 10 is an image from the same data set showing an internal view, making the shell structure more apparent. This view also demonstrates one of the strengths of volume visualization, the ability to view the internal and external structure of a data set simultaneously.

Figure 11 illustrates the same features present in the spheres image for a cylindrical data set. The cylindrical data was constructed in a cylindrical coordinate system where:

$$x = r\cos(\theta) \quad (10)$$

$$y = r\sin(\theta) \quad (11)$$

$$z = z \quad (12)$$

and :

$$0 \leq \theta \leq 2\pi \quad (13)$$

As in the case of the spheres, the radius, r, was again increased by units of one starting at one to obtain several cylindrical shells. The lower photo in Figure 11 is an image of the same data set in with a portion of the image carved away to reveal the internal structure. As in the case for the spheres, the same data values were used in order to make direct comparisons between the two figures.

The last data set fabricated for use in the first step of the visual validation process was a series of truncated conic shells. The truncated cones were also built in a cylindrical coordinate system with the only difference being that the radius varied

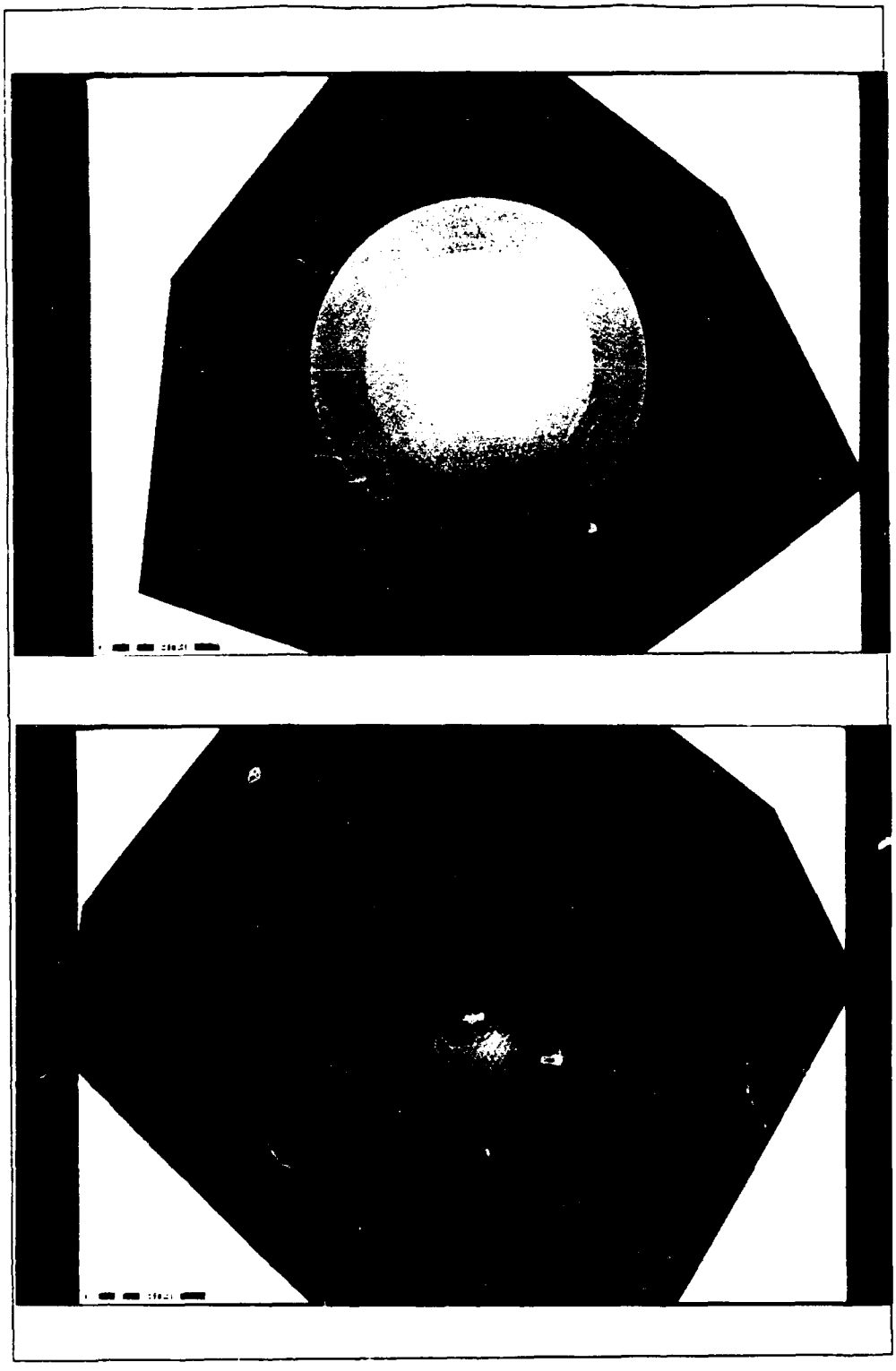

Figure 10. Concentric Spheres, External (Upper) and Internal (Lower) Views

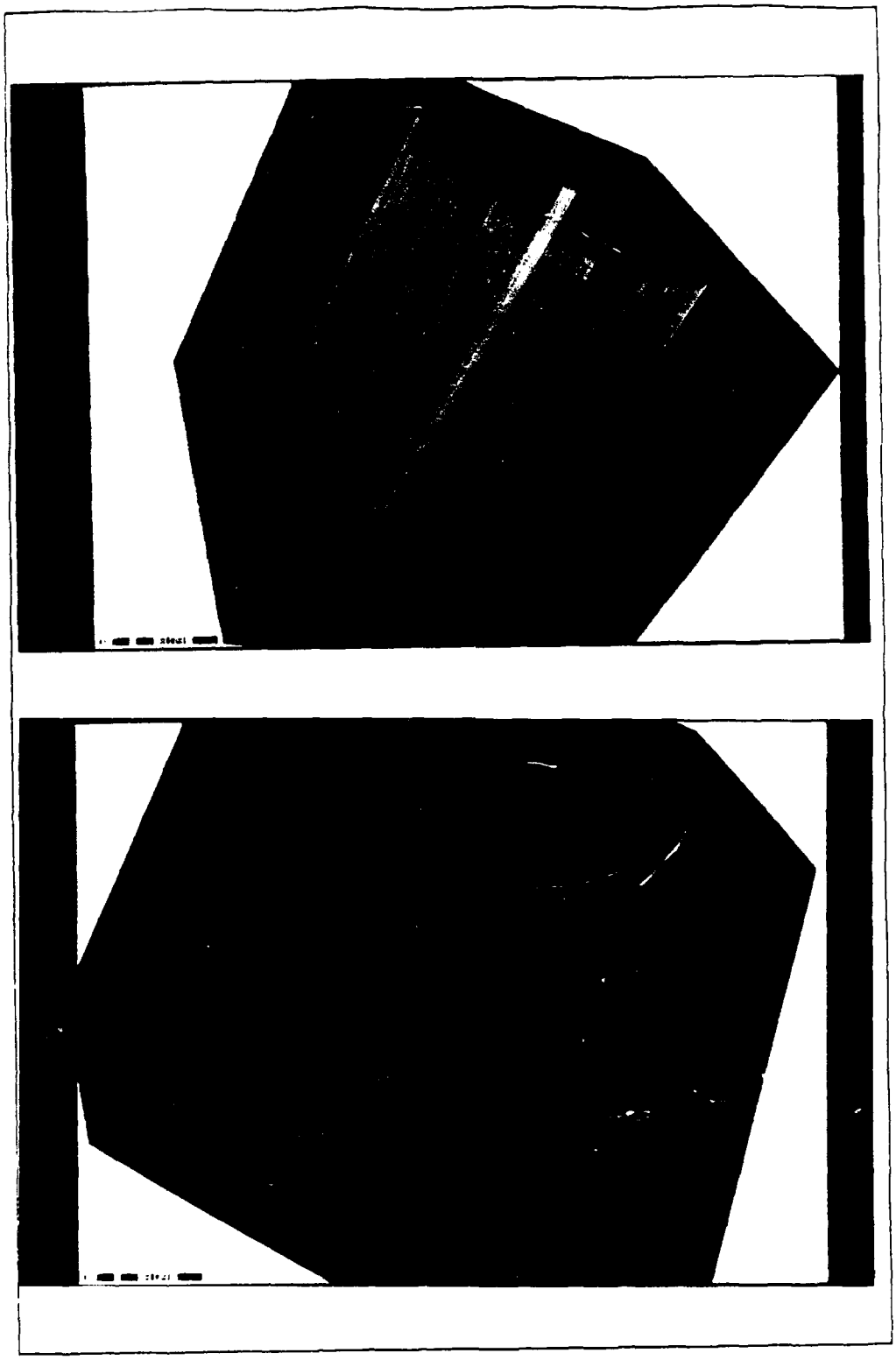

Figure 11. Cylindrical Shells, External (Upper) and Internal (Lower) Views

linearly as a function of z. Figure 12 contains photos of the outside and internal views of the conic shells.

The images from the fabricated data sets indicate that the structure of the data was maintained during the sampling process. In order to provide further confidence, test data from a finite–element analysis were obtained, and images of that data were also rendered.

Visual Validation Using Actual Simulation Data. For the final step in the visual validation process, test data from a finite–element analysis was obtained from Dr Phil Webste- at the Wright Research and Development Center at Wright–Patterson Air Force Base (26). Two data sets were selected, and several images were rendered from each set. The images were presented to Dr Webster for his qualitative comments. In addition, the images from each data set were compared to images generated using PLOT3D (25). PLOT3D was chosen because it is the tool used by researchers in computational fluid dynamics.

PLOT3D uses polygonal prin itives to render images from data on curvilinear grids. As pointed out in Chapter 2, the dependency on polygonal primitives limits the ability of the rendering software to viewing the external structure of the volume data. This would not be the case if PLOT3D utilized transparent polygons. Some versions of PLOT3D have the capability of using transparent polygons, but that capability is limited. This makes direct comparisons of images from the two renderi g methods difficult. It was possible, however, to compare the structure of several Mach number shells in the data with images rendered using PLOT3D.

Figure 13 is an image from the first data set where the Mach number is equal to 1. The upper photo is the image produced by PLOT3D, and the lower photo is the image produced by the mapping program and VIPER. The overall structure of the Mach shell in both images "agree well". The subtle differences are due to a combination of the different rendering methods, and the sampling problem mentioned earlier.

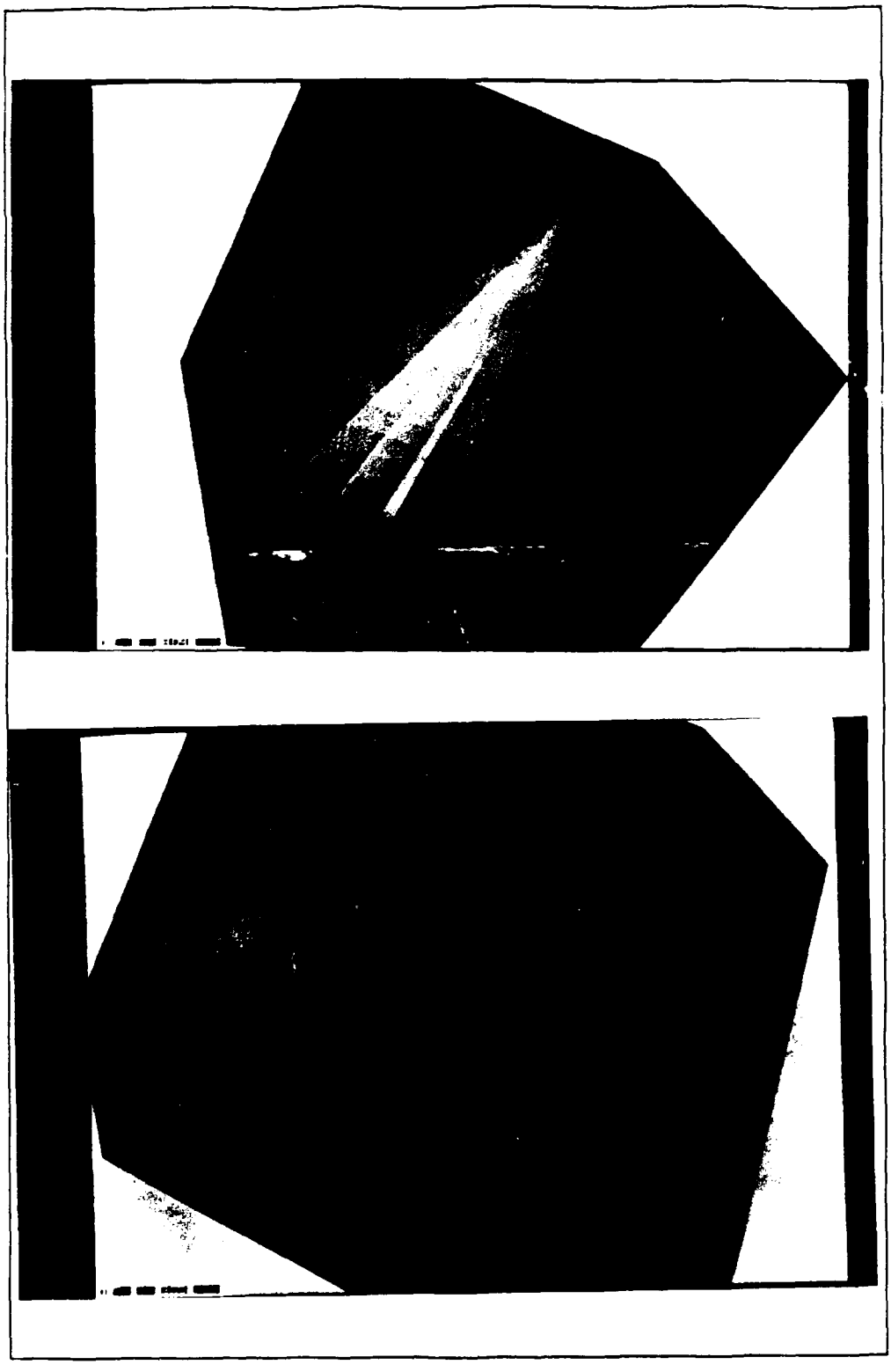

Figure 12. Conic Shells, External (Upper) and Internal (Lower) Views

Figures 14 and 15 are images from both PLOT3D and VIPER using different Mach numbers. In Figure 15 the *wing–like* structures are actually the representation of the "tunnels" where the Mach number is equal to 1.9. A close examination of the VIPER image reveals the same structure is present, but is represented differently because of VIPER's non–dependency on polygonal primitives.

In all cases Dr Webster (26) believed the VIPER images were reasonably accurate representations of his data. The remaining figures in this chapter are images generated using the same data values from for second data set. Again, the images from VIPER agree well with those from PLOT3D in terms of the structure of the Mach shells.

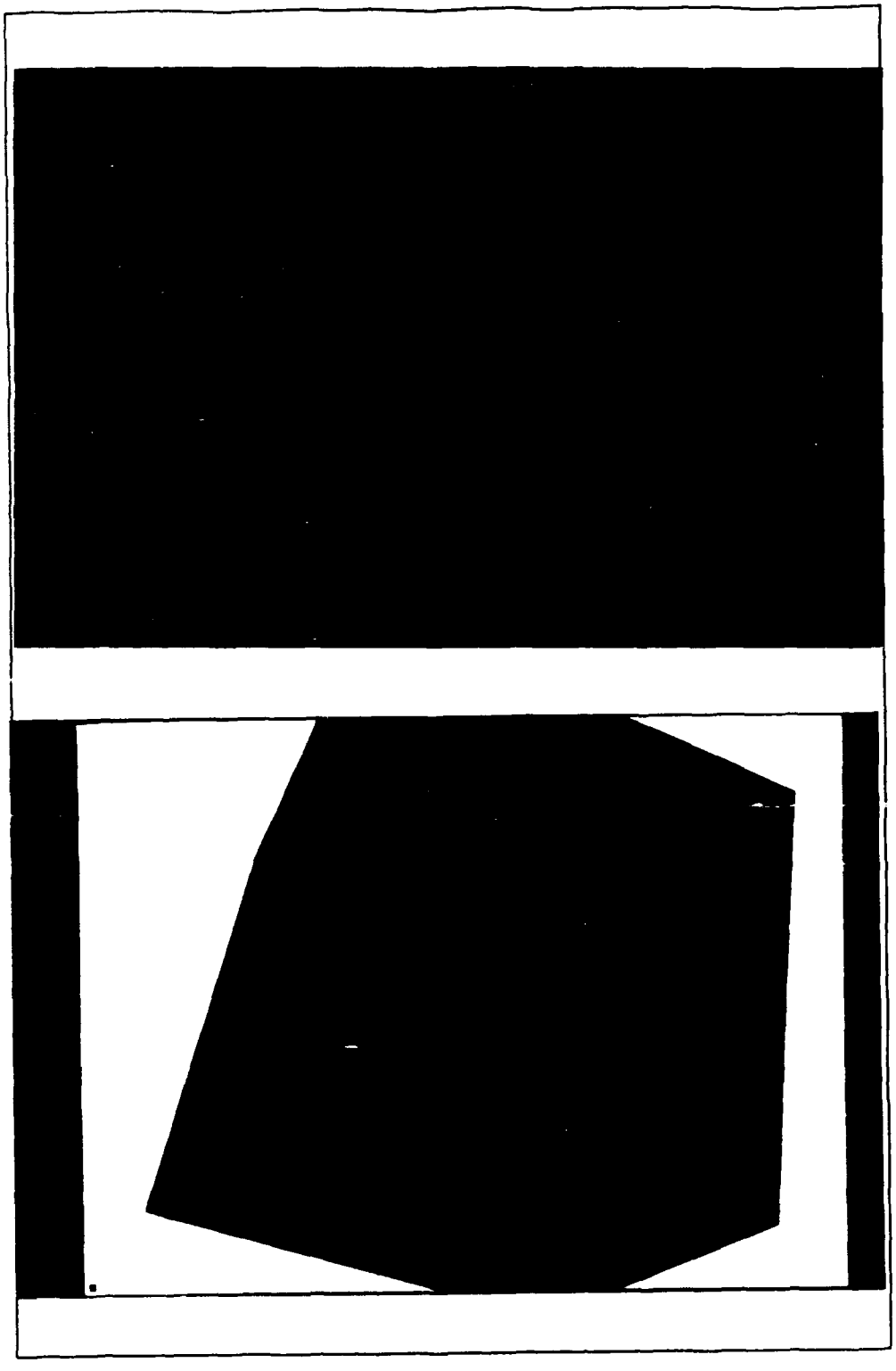

Figure 13. The Mach 1.0 Shell From Data Set #1 (PLOT3D Upper, VIPER Lower)

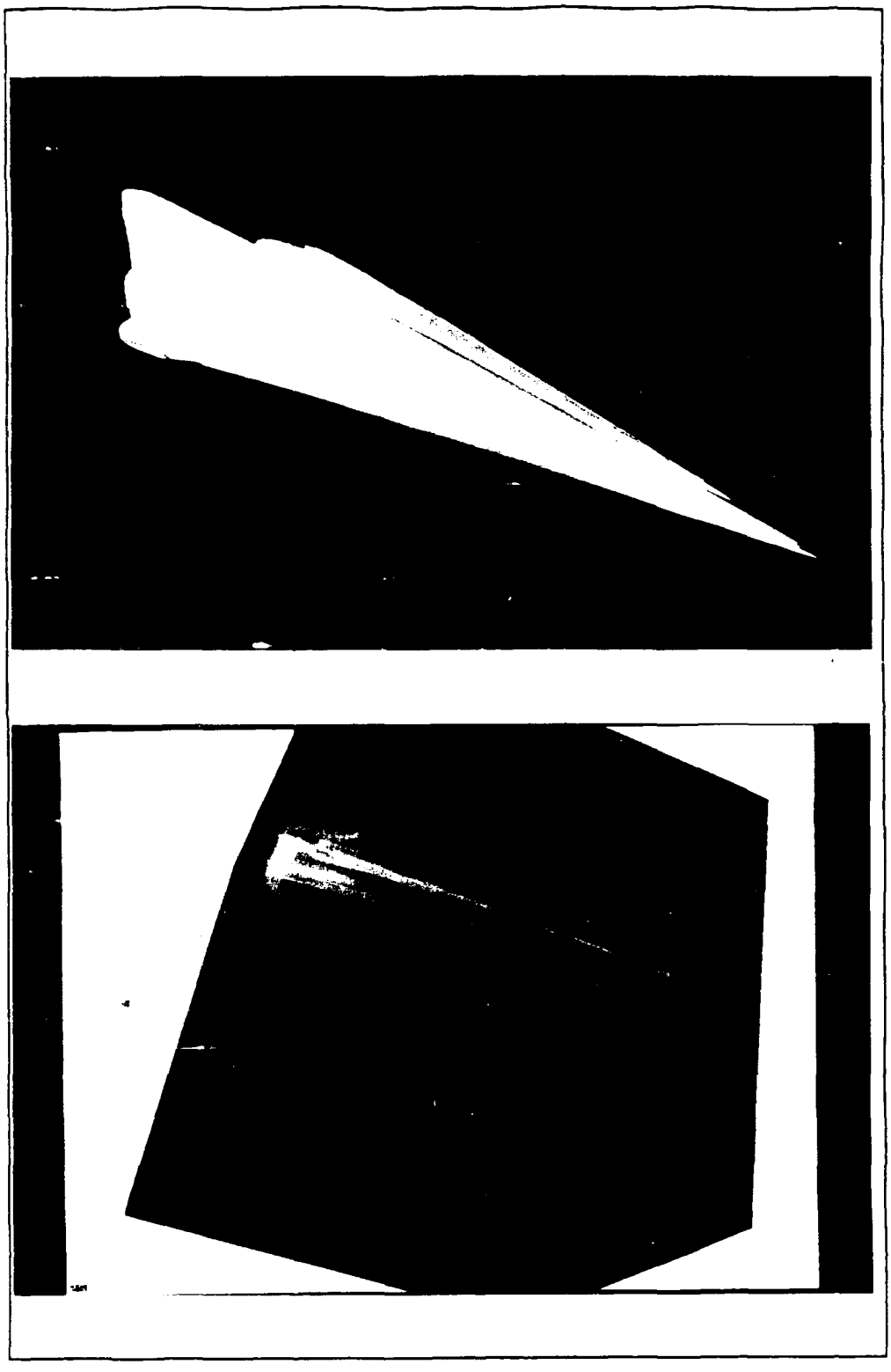

Figure 14. The Mach 2.5 Shell From Data Set #1 (PLOT3D Upper, VIPER lower)

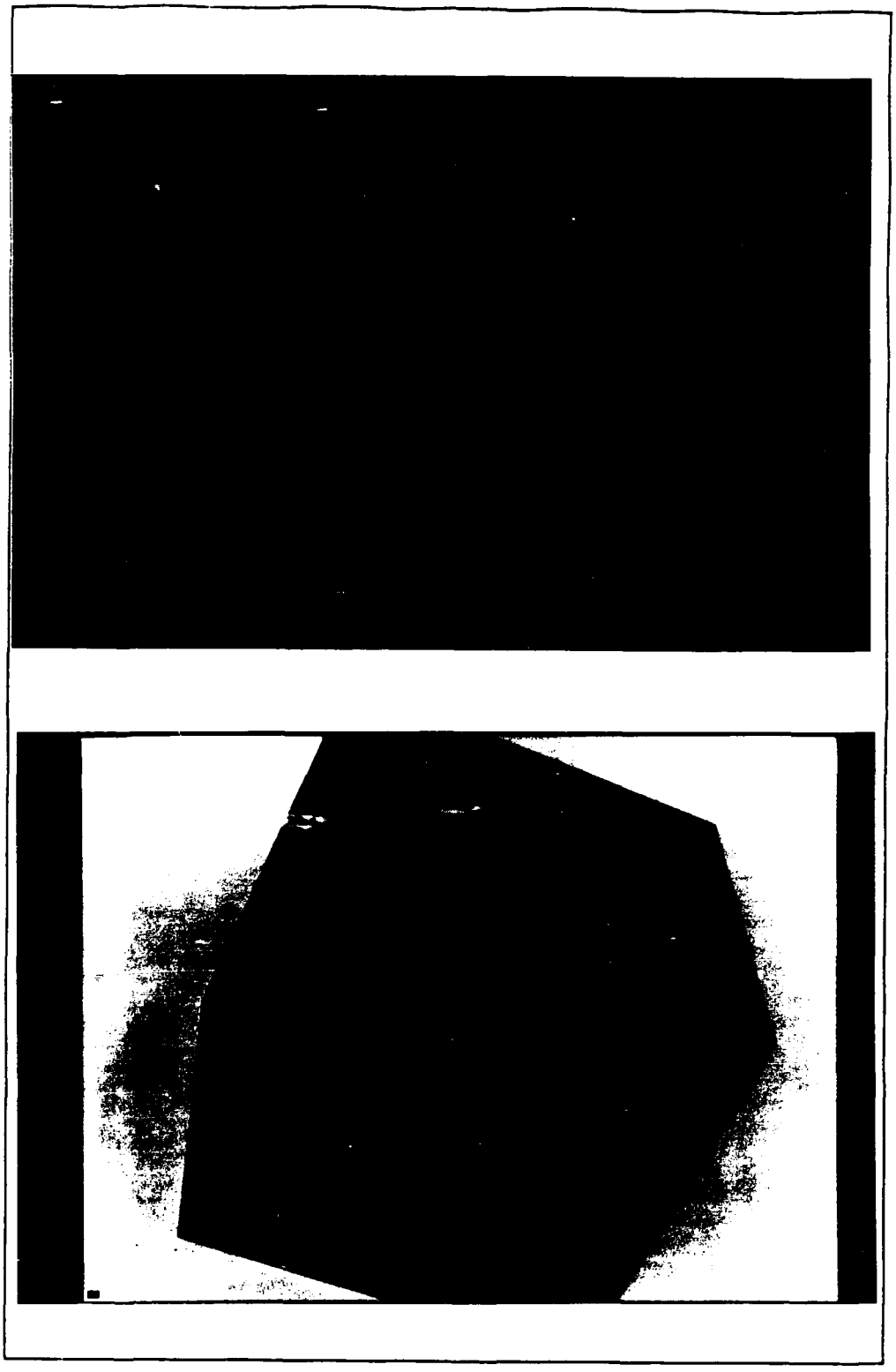

Figure 15. The Mach 1.9 Shell From Data Set #1 (PLOT3D Upper, VIPER lower)

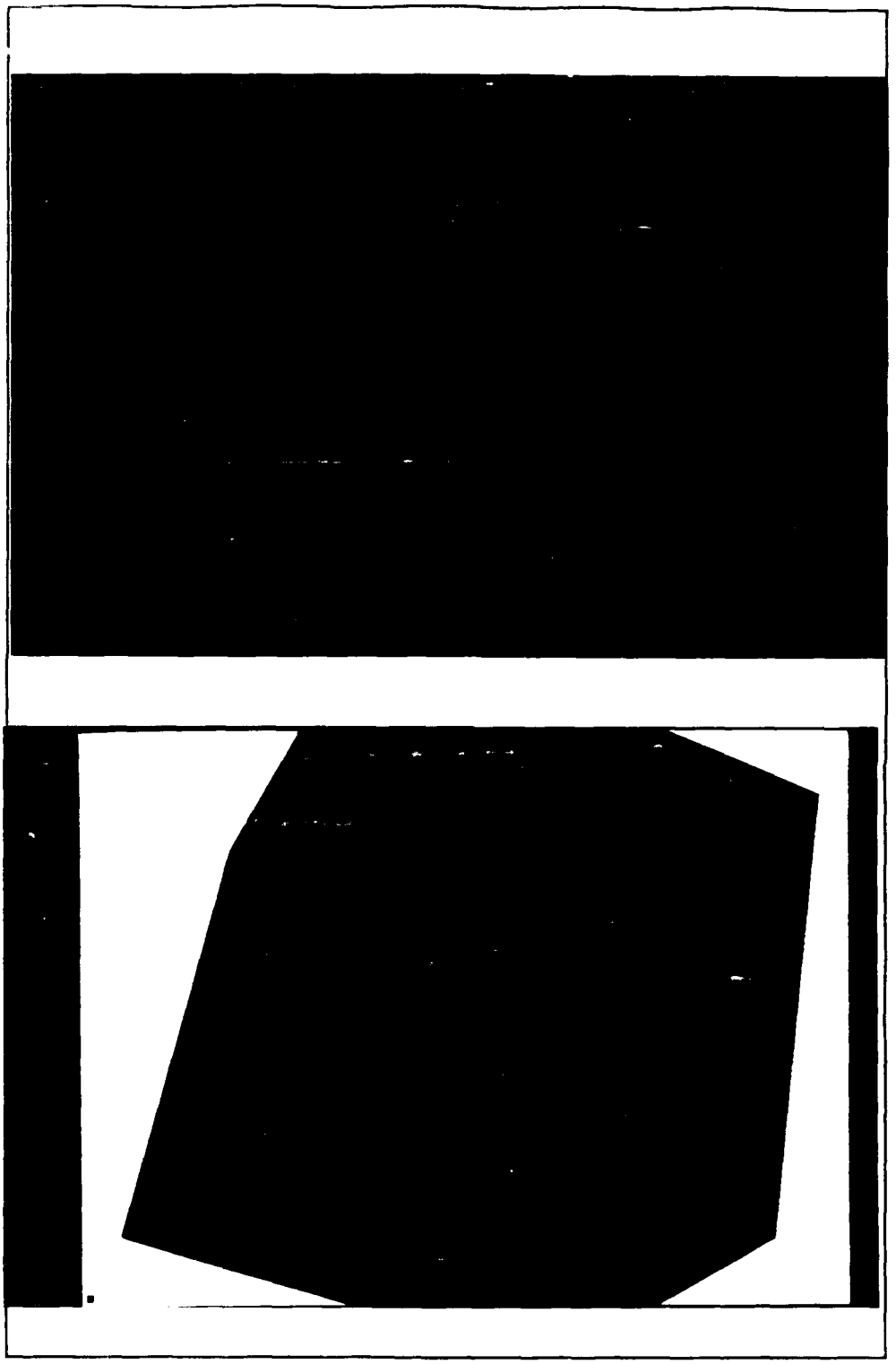

Figure 16. The Mach 1.0 Shell From Data Set #2 (PLOT3D Upper, VIPER Lower)

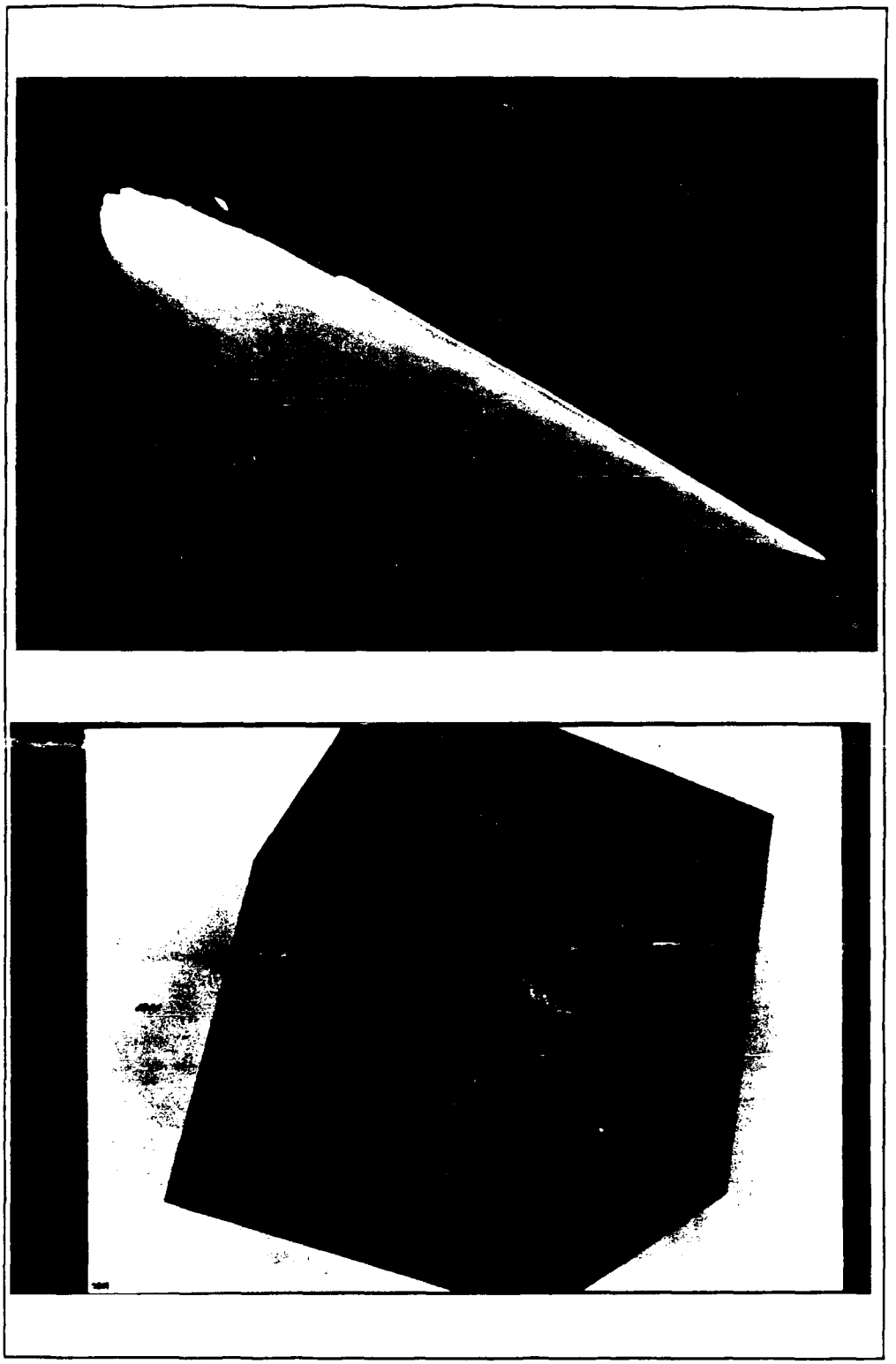

Figure 17. The Mach 2.5 Shell From Data Set #2 (PLOT3D Upper, VIPER Lower)

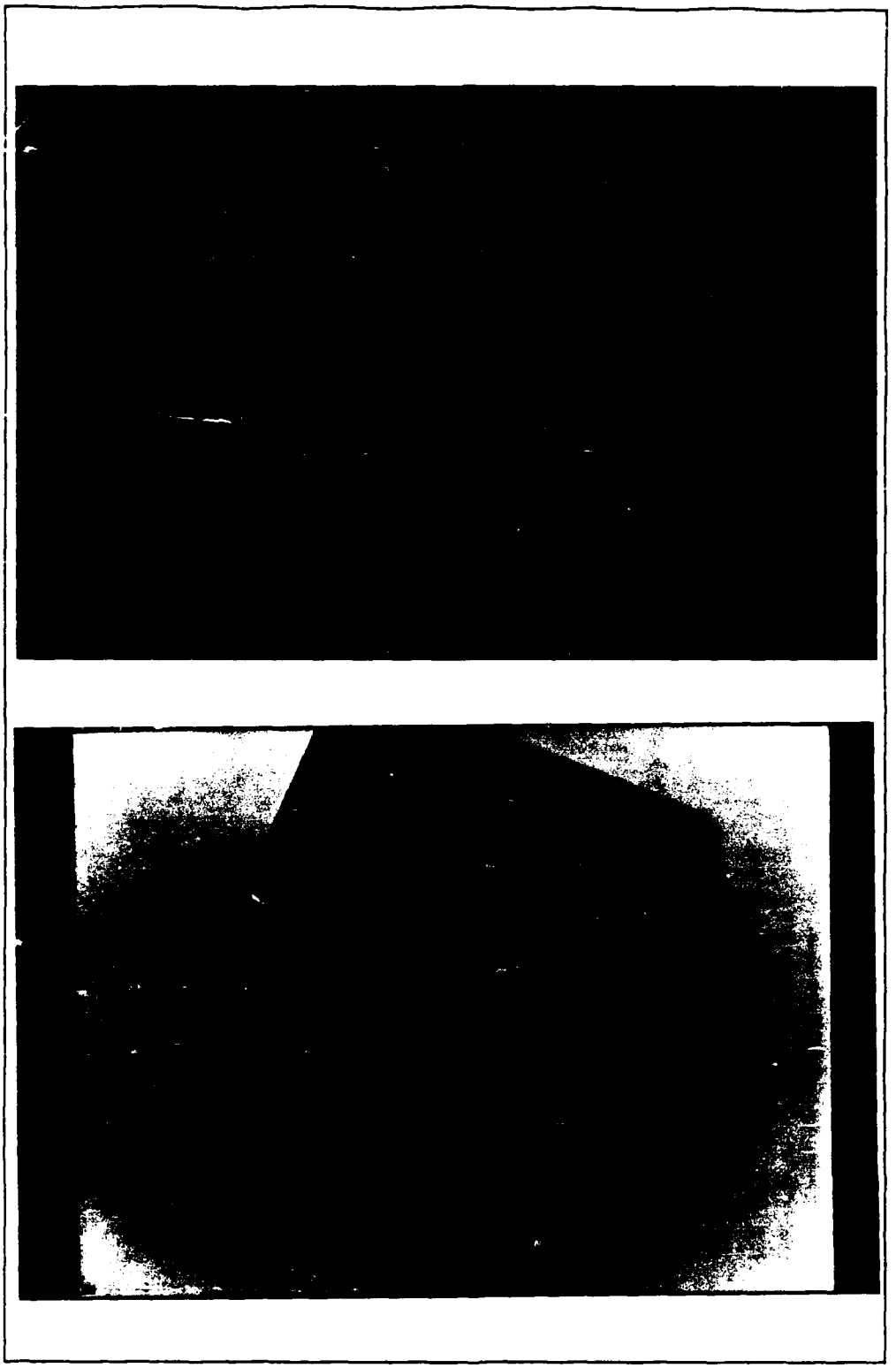

Figure 18. The Mach 1.9 Shell From Data Set #2 (PLOT3D Upper, VIPER Lower)

5.5 *Test Results From Varying the Sampling Parameters*

The two sampling methods produced images that varied somewhat. This was not surprising considering the differences in the two methods. The A–buffer method for sampling produced better images than the Z–buffer method.

A–Buffer Sampling. The A–buffer method for sampling produced more accurate images from the data using both combinations of grid construction techniques. The images of the simple geometric shapes presented earlier were rendered using A–buffer sampling with the Gaussian weighting filter. The data structure was maintained, and data accuracy was reasonably maintained as well. Table 3 lists the Mach numbers at several of the shells along with the target values, ranges, and shell colors given to VIPER during the rendering process. As the table shows, data accuracy was maintained to within 0.07 units.

The distance weighting filter used with the A–buffer sampling was able to preserve the data structure, but did not perform as well as the Gaussian filter for preserving accuracy. The distance weighting filter assumes a constant, second order distribution of the weighting terms based on the filter radius. If the frequency at which the data is changing is high enough, this filter under samples in some areas. The only way to increase the sampling rate using this filter is to shorten the radius. The dependency of the filter radius on the data distribution is clearly shown in Figure 12. The inner and outer most shells disappear toward the top of the image. This is the result of the variable density of the data.

In each of the images, the data points lie at constant intervals along the radius. For the sphere, this is true for any radius line, but for the cones and cylinders, it is true only for radius lines which lie in a single z plane. As the data is viewed along constant lines of radius, while varying ϕ or θ, the density is inversely proportional to the length of the radius. Therefore, at the outer limits of the data set, the filter radius would not be large enough to properly sample the data using a simple

Table 3. Mach Number Values for Fabricated Data Sets

Shell Number	Mach Number	Target Value and Range	Color
1	0.00000000		
2	0.00000000		
3	0.00000000	0.00 ± 0.01	white
4	23.14550249		
⋮	⋮		
20	5.61360891		
21	5.45544726	5.40 ± 0.07	blue
22	5.30994244		
⋮	⋮		
39	3.85758375		
40	3.80509717	3.80 ± 0.01	green
41	3.75469631		
⋮	⋮		
58	3.12093892		
59	3.09294787	3.09 ± 0.005	yellow
60	3.06569670		
⋮	⋮		
78	2.67261242		
79	2.65497122	2.65 ± 0.01	red
80	2.63767481		

density weighting filter. This observation supports the idea that the filter must be dynamically determined based on the data distribution at each sample point.

Determining the filter parameters should not be difficult, because the density and frequency of change are related. The density can be determined by indexing the curvilinear grid at each sample point to calculate distances to the nearest neighboring data locations. It is safe to assume that in areas of close grid spacing the probability is high that the data may have a greater frequency of change (21). Further investigation to determine the functions used to set the filter parameters is required.

With a Gaussian filter, the sampling rate can be altered without changing the

filter radius. By specifying the number of standard deviations spanning the radius, the sampling frequency can be changed without altering the radius. Unfortunately, it is still necessary to determine the distance of neighboring data points to determine the data frequency. The advantage of the Gaussian filter over the distance weighting filter is that one additional degree of freedom is provided. Functions for determining both the radius, and the number of standard deviations offer a wider range of control over the filter characteristics.

Grid Construction. In general, the method of constant plane spacing resulted in more accurate images than did the variable spacing method. The variable spacing method required an order of magnitude more CPU time to sample the data over the constant spacing method. This was due primarily to maintaining linked lists of unique coordinate values.

Figure 15 was rendered using constant plane spacing in the rectangular grid. Figure 19 is the same data rendered from a variable spaced grid. Comparison of the two images reveals the variable spaced grid sampled the data well in areas where the planes were closely located to the data points, but poorly in other areas.

The algorithm implemented for determining plane location placed too much emphasis on the dense areas of the input data. Information from the less dense areas was lost, and the grid contained cells with extreme size variances. The variable plane spacing method may still have some merit, but it should be modified to spread the plane locations using less emphasis on the dense areas of the input data.

These results demonstrate that spreading the sample locations over a wider area produce better results than attempting to sample only in very dense areas. A smarter algorithm for spreading the location of the planes could possibly result in better sampling, but that algorithm may be very difficult to implement, and it is doubtful that the increase in sampling accuracy would out weigh the additional processing time. The second method used for sampling was the Z-buffer technique. The results for this method were similar for the two grid construction methods.

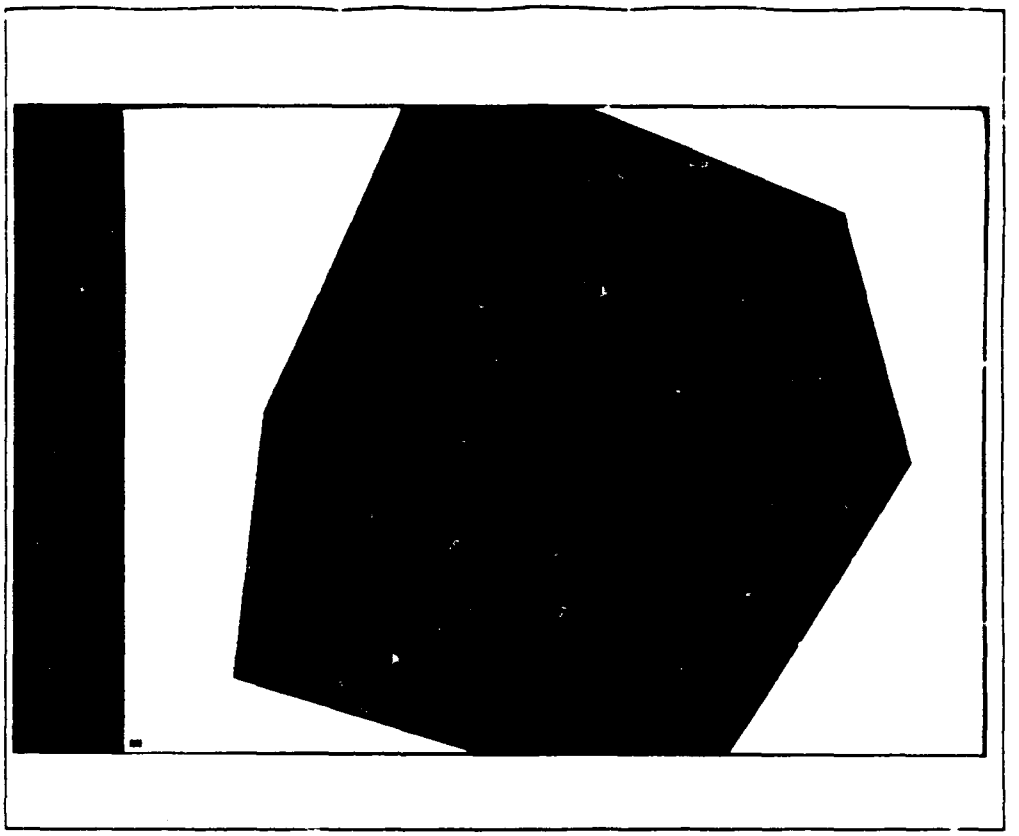

Figure 19. A Mach 1.9 Shell From Data Set #1 Using Variable Plane Spacing

Z–Buffer Sampling. The Z–buffer sampling method required about half the time to e .ecute than the A–buffer. This, however, was the only principle advantage to this method. Figure 20 is an image of the Mach 1.9 shell from Dr Webster's first data set using Z–buffer sampling with constant spaced planes. Figure 21 is the same data, and sampling method, using variable spaced planes. Both figures show that many cells within the grid contain constant color, and have a mosaic appearance.

This is characteristic of Z–buffering techniques. The algorithm simply places each data point at the closest sample location it finds. The same sampling problems occurred with the regular and the variable spaced grids. Although a noticeable difference in image quality is apparent, the Z–buffer sampling method still has some utility for data visualization.

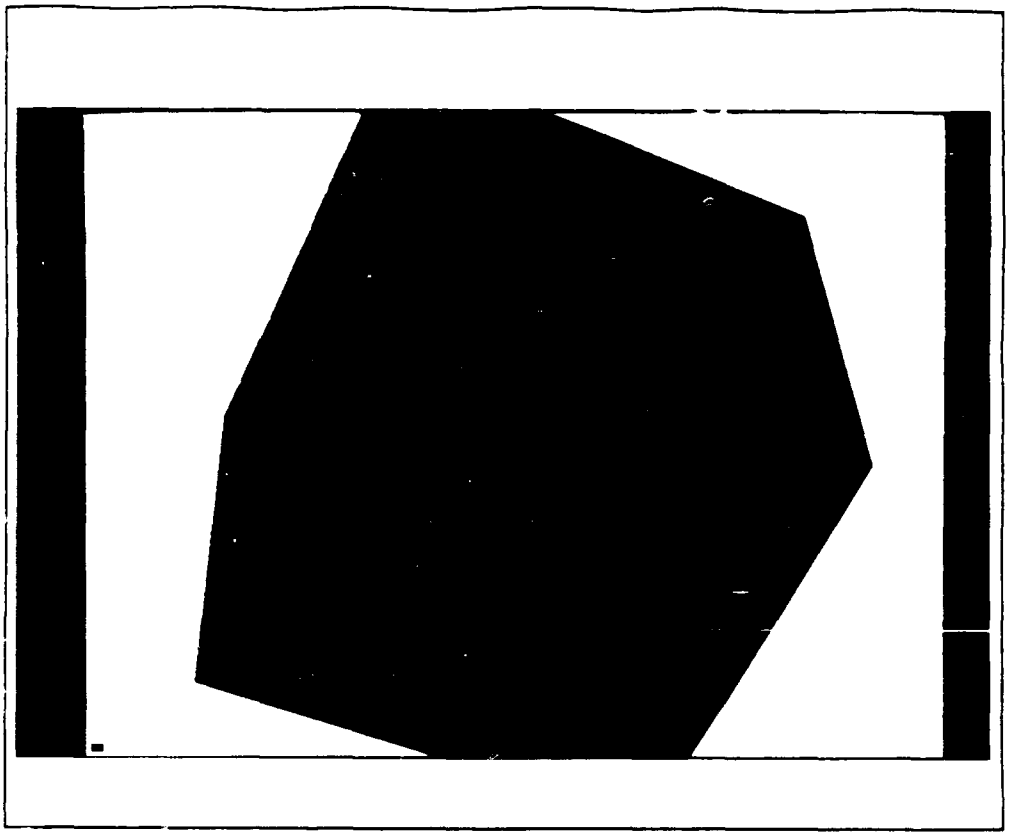

Figure 20. A Z-Buffer Mach 1.9 Shell From Data Set #1 With Constant Plane Spacing

Because of the faster execution time, the Z-buffer method can be used for preliminary visual analysis. This point illustrates the flexibility the mapping program offers. The program allows users to trade computation time for accuracte data representation. The Z-buffer sampling method also helped in determining the resolution of the rectangular grid.

The Z-buffer placed data samples only in the rectangular grid cells where an input data point was located. If the rectangular grid was constructed at too high a resolution, many grid cells remained empty. The resultant image appeared broken up in slices along any axis where the resolution was too high. Repeated experimentation with the Z buffer helped determine the resolution to build the rectangular grid.

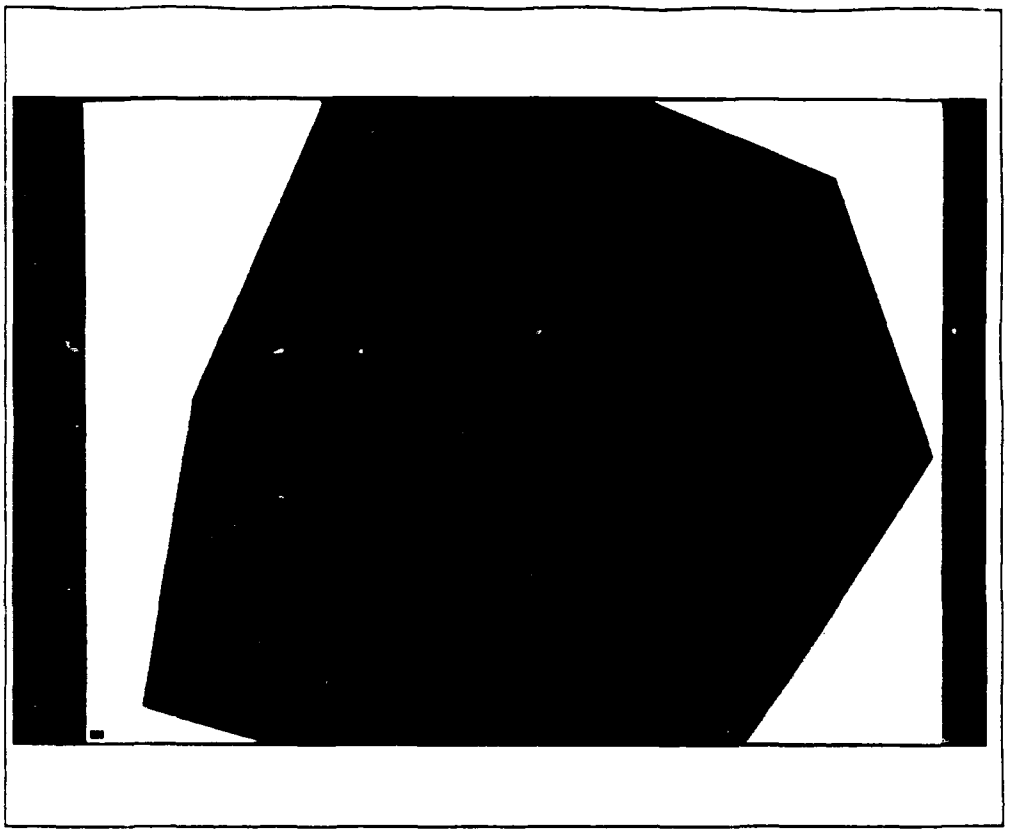

Figure 21. A Z-Buffer Mach 1.9 Shell From Data Set #1 With Variable Plane Spacing

The results of the program testing show the concept behind sampling data onto rectangular grids was successful. With continued research in this area, it is likely image quality can be further increased. The next chapter presents conclusions, and suggested areas for continued research.

VI. Conclusions and Recommendations

6.1 Introduction

The conclusions from the work accomplished are presented in this chapter. The success of the project is evaluated based on those conclusions. Potential problems are identified, and possible solutions are recommended. Finally, suggested areas of continued investigation are presented for both the sampling program and the rendering software.

6.2 Conclusions

The results of this work indicate it is possible to sample data from an arbitrary source onto a rectangular grid for volume visualization. The images of the simple geometric shapes presented in Chapter 5 clearly show that the structure of the input data can easily be preserved during the sampling process. It is not, however, as obvious that data accuracy can be maintained during the sampling process.

Sampling Problems. All the images presented in Chapter 5 show artifacts due to sampling problems. The sampling problems occur when the data is mapped to a rectangular grid, and when the images are rendered. As Craig Upson (24:27) points out, three–dimensional techniques for interpolation, or other sampling methods, do not satisfy the requirements for visualization. He says one more dimension is required, and adds that the details of how to achieve that are only just beginning to be worked out. Both the rendering software and the sampling program contributed to data inaccuracy.

Image Rendering Sampling Problems. The lines forming rectangles in the images are a result of VIPER's rendering method. VIPER linearly interpolates data values from each vertex in a cell to find the data value at the entry and exit points of the cell. Along the edges of any cell, linear interpolation results in a constant

variation of the data between only two values; the two on either end of the cell edge. If those two values lie within the range of a target value, every point along that edge has the same color assignment.

This results in visible lines of constant color which stand out in the image, because at small distances in any direction away from the edge, the color begins to vary as a function of more than two data points. This can also occur when a sample ray lies completely in the plane of one of the cell faces. When this occurs, the four data points at each vertex of the cell face completely determine the sample values along that ray. If those four values do not vary much, or completely fall in the range of a target value, the same phenomenon occurs. This problem is related to using only one sample ray per pixel to determine color.

A possible solution to this problem would be to sample every pixel with more than one ray, and average the samples. This form of sampling is recognized as one of the two accepted methods for reducing artifacts due to under sampling (13:33). The other accepted method is a convolution technique where the final pixel color is determined by a weighted sum of the colors from neighboring pixels (13:33).

The increased sampling method directly affects the time required to render an image. In fact, the increase is linearly proportional to the number of samples. This is because the same number of interpolations are required for each sample. The convolution method, however, can be applied after image generation.

The advantage to performing the convolution after image generation is a savings in CPU time, and the opportunity to use several filters on each image. The convolution technique requires fewer calculations to weight the color values of each pixel. In addition to the CPU time saved, once an image is created, any number of convolution filters can be applied to that image. Another alternative for reducing the sampling artifacts due to image rendering would be to change the rendering algorithm.

A cell integrating method can be implemented to render the data images. Once the cell is integrated, every pixel covered by the cell's projection onto the viewing plane can be processed. This technique results in images with far superior quality, but is not recommended. As stated in Chapter 2, the ray casting method is much faster than a cell integration method. During this project, it was observed that VIPER required between two and twelve CPU hours to render an image using a Sun 4. The time was dependent on the grid size, and the image size.

The excessive processing time caused problems in experimentation with different sampling parameters. If a cell integration method were implemented, the only benefit would be higher quality images. What is needed during a research project is a fast method for image rendering to determine the effect of varying the control parameters. The data mapping program also experienced sampling problems, but these problems had a different effect on the resulting images.

Data Mapping Sampling Problems. Sampling problems in the data mapping program were related more to maintaining the accuracy of the input data. Curvilinear grids used for finite–element analysis can have a wide range of scales. This range of scales causes a corresponding range of spatial densities of the data.

The mapping program was designed with a constant filter radius which the user can set. The problem with using a constant radius is that the Niquist criteria can be violated. Sampling with a constant filter size assumes an input signal with a frequency range corresponding to that filter. It was very difficult to find a radius which did a good job sampling the data from every part of the input grid. If the radius was set too high, the outer sections of the grid were sampled well, but the dense areas were under sampled. The opposite was true for small filter radii. This means the input signal frequency range was much broader than anticipated.

This resulted in a loss of data accuracy, which was difficult to control and identify. A possible solution would be to dynamically determine the search radius for every section of the input grid. The grids are easily indexed using the i, j, and

k index variables, and the relative grid spacing can be determined at every data sample.

If the relative grid spacing is determined for each data sample, the filter radius, and number of standard deviations could be determined based on the spacing. It is safe to assume that in sections of a curvilinear grid where the spacing is very small, the frequency at which the data could be changing might be much higher than in other areas (21). The radius should be set much smaller, and the number of standard deviations much higher, when sampling data in these areas.

Implementing a dynamic filter radius should not slow down the program significantly. The program executes on a Sun 4 in between 45 minutes and two hours of CPU time for an input grid of size $80 \times 80 \times 80$. Finding the functions needed to determine that radius may require considerable investigation. The radius size should be determined from both the input data, and the rectangular grid.

Even with the problems presented, the project was still successful. The results clearly show that data structure can be maintained. The images rendered from the test data donated by Dr Webster (26) also show that reasonable data accuracy was maintained, and good agreement with PLOT3D (25) was achieved. These promising results justify continuation of research in this area.

6.3 Recommendations

Recommendations for future research efforts in the area of volume visualization are presented in this section. Specifically, recommendations for enhancements to both the rendering software and the sampling program are proposed. Several of the proposals focus on new concepts which should be considered, while others recommend methods to improve existing capabilities.

Future Enhancements. Volume visualization is still a relatively new concept in computer graphics for analyzing three–dimensional data. Much emphasis on the importance of visual analysis has been expressed by the scientific community.

In the months spanning this research effort, articles on volume visualization have appeared almost monthly in publications such as *IEEE Spectrum*, *IEEE Computer Graphics and Applications*, and *IEEE Computer*, to name a few. The Association for Computing Machinery annual conference on computer graphics held in the summer of 1989 had a significant portion of their presentation dedicated to the subject of data visualization.

The Department of Defense has also expressed interest in data visualization (15). Because the concept is still relatively new, there are many unanswered questions, and many techniques that have not yet been considered. According to (17), the human brain is is capable of processing approximately 40 to 50 bits of information per second when reading, or looking at pictures. Volume visualization can be viewed as a means to reduce the size of a data set in an attempt to find those 40 to 50 bits which adequately convey the information. For these reasons, continuing research into the subject of data visualization is justified.

Viper Enhancements. VIPER currently renders images using only one sample per screen pixel to determine color. This under sampling causes artifacts in the images for reasons previously discussed. VIPER should be enhanced to include both methods previously mentioned for aliasing reduction. This improvement will provide higher quality images without completely sacrificing the rendering time advantage over cell integration methods.

Another weakness in the program is the requirement placed on the user to select target values for image rendering. If some peculiar phenomenon is occurring in a data set, it may go unnoticed if the user does not happen to select the correct target values for viewing. A method could be implemented where the range of the data is found, a color table created, and all the information in the data set is rendered in the image. If a color legend is created showing which data values correspond to each color, users can identify strange phenomenon, and associate it with a data value.

Finally, VIPER should not be limited to rendering images from a single scalar

value. In a computational fluid flow simulation five data values are associated with each grid point. These data values can be used to calculate several variables of interest. For a 100 × 100 × 100 grid, VIPER uses approximately eight megaBytes of memory during execution. This usage is well below the abilities of many scientific workstations, and should be exploited. The grid data structure can be easily modified to retain more than a single data value for each grid node point. This would allow users to render images with several different data values present simultaneously. The interaction of several variables could be examined to determine their behavior with respect to one another.

Mapping Program Enhancements. The results of this project have demonstrated that sampling data onto rectangular grids is possible and feasible. Several problems associated with sampling were discovered during the course of the investigation. The dynamic filter radius is certainly the first area for further research to consider. The code required to implement a dynamic filter should be straight forward, relatively easy to implement, and have a minimal impact on the program's performance.

Different filtering functions should also be considered. Further investigation into the characteristics of the data generated through finite–element analysis should be accomplished to determine if the data follows some known analytic, or approximate, mathematical distribution. Results of this investigation could lead to the implementation of several filtering functions, offering users a choice to match the filter as close as possible to the behavior of the data.

According to Dr Webster (26), a phenomenon of particular interest to fluid flow researchers is the rate of change of Mach number values in their simulations. To derive the surface gradients would require a program of considerable complexity, but it would have benefits for the rendering program. In the areas of constant Mach values, the rate of change tends to be very slow (26). This causes extremely thick surfaces of constant values, and makes transparent viewing of the images difficult.

The gradients of these surfaces, on the other hand, should be very sharp, and result in much thinner surfaces making transparent viewing much easier.

Finally, the sampling program implemented the ability to calculate Mach number, and pressure number values only. Other values of interest, such as vorticity can be calculated from the data on the input grids. Further investigation is required to determine which values of interest should be offered by the program.

Appendix A. *Requirements Analysis*

This appendix contains the formal requirements specifications using the $IDEF_0$ requirements specification language. For an explanation of the $IDEF_0$ language, see Section 3.2. References for the specification language are also given in that section.

Each box in every diagram represents a required function in the mapping system at varying levels of abstraction. The facing page text for each diagram provides an explanation for every function in the diagram. Every diagram, except the first, has a facing page text document associated with it. In each case, the facing page text precedes its associated diagram.

The diagrams progress from a very high level of abstraction to increasingly lower levels, identifying the requirements for the mapping system at each level. The mapping program is not an exceedingly complex software package. This is reflected by being able to completely specify the requirements in only two levels of abstraction using $IDEF_0$.

The results of the requirements analysis were used to implement the detailed design for the mapping system. Appendix B contains the structure charts used to accomplish the detailed design. The detailed design is meant to have a very close relationship with the requirements analysis. In fact, there should exist very close to a one–to–one mapping between the requirements analysis and the detailed design. The detailed design will, of course, contain more detail than the requirements analysis, but the reader should be able to follow the development of the design from the requirements analysis.

A-0 Map Data To Rectangular Grids

Abstract: This system accepts as input data from an arbitrarily shaped grid, or scattered data. The data is sampled onto a rectangular grid, and a user file is written to the disk.

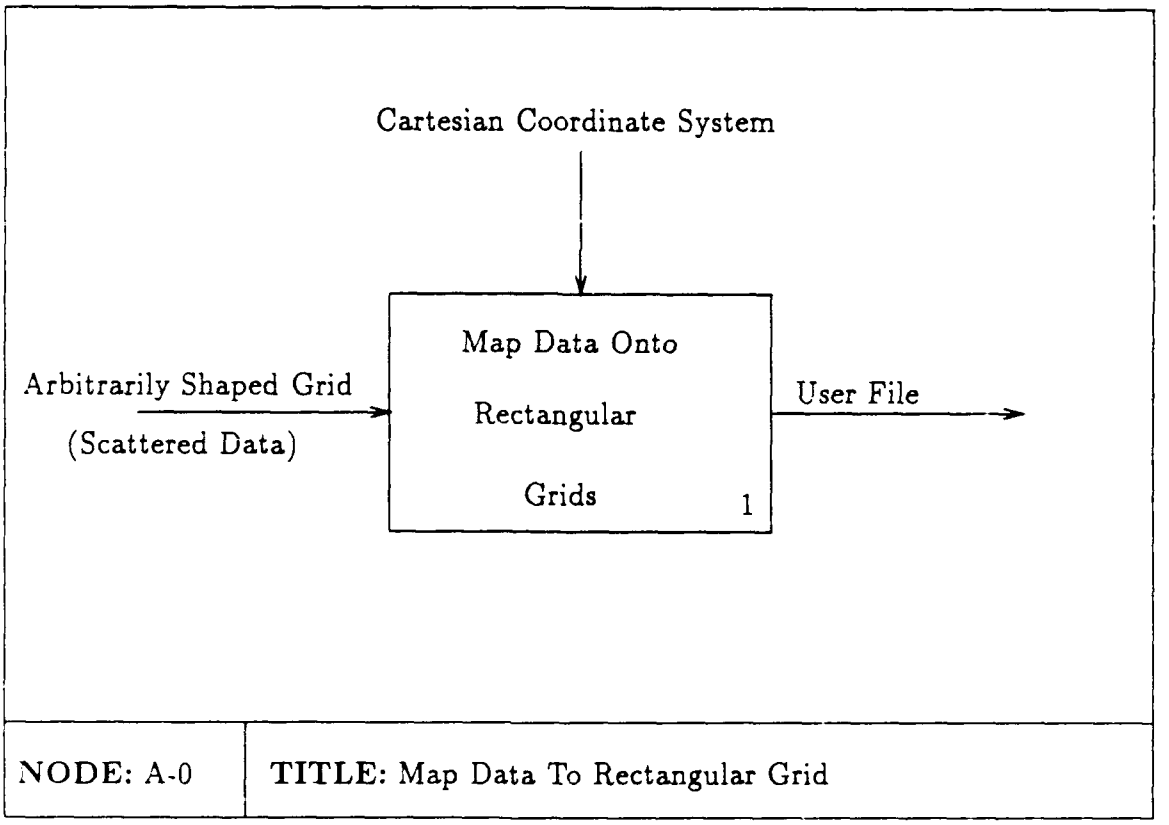

Figure 22. $IDEF_0$ A-0 Diagram

A0 Map Data to Rectangular Grid

Abstract: This system samples data from three dimensional, arbitrarily shaped, data grids, or scattered data, onto a rectangular grid.

A1 Initialize System: This activity reads the user parameter file and configures the system according to his/her desires. Table 2 describes each parameter under the user's control.

A2 Process Grid: This activity takes an arbitrarily shaped grid as input, and processes it into a rectangular format suitable for the image rendering software.

A3 Save To Disk File: This activity saves the transformed data grid to a disk file in the format required by VIPER. The file name the grid is saved under is specified by the user.

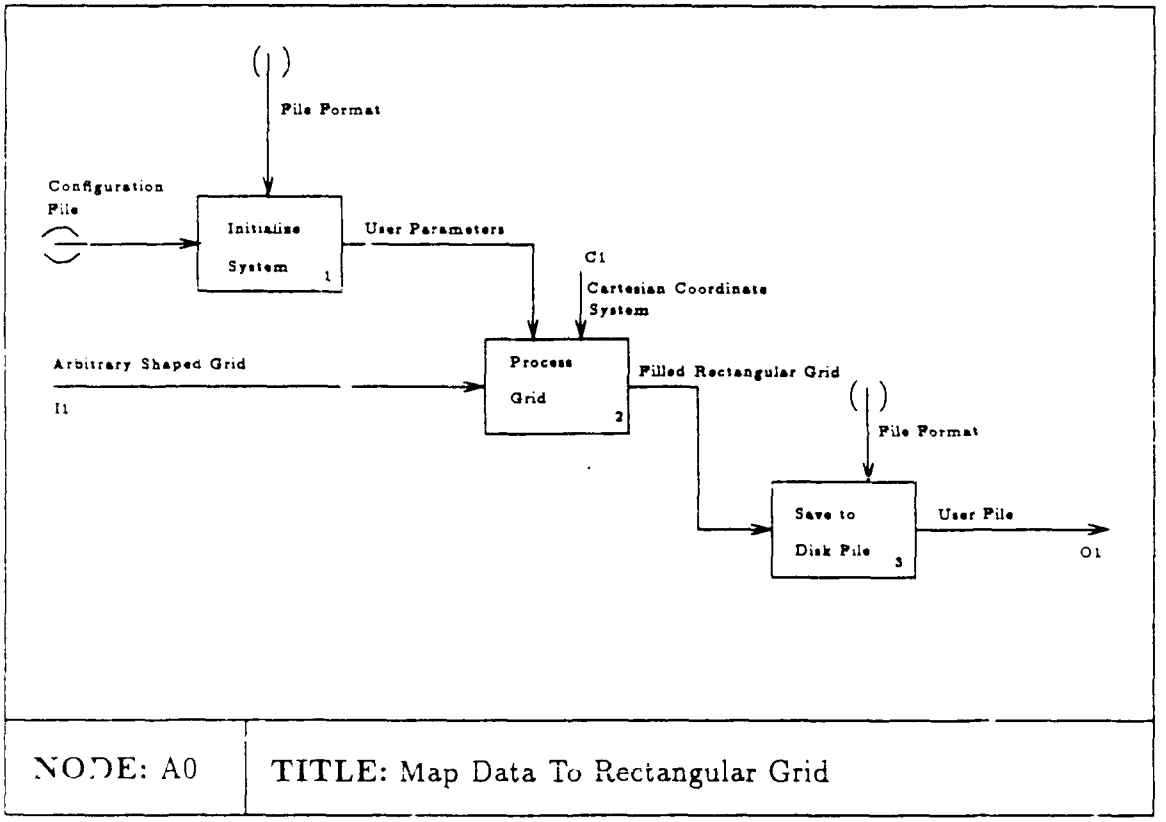

Figure 23. IDEF$_0$ A0 Diagram

A1 Initialize System

Abstract: This activity reads the user parameter file and configures the system according to his/her desires. Table 2 describes each parameter under the user's control.

A11 Read Input File: This activity reads the parameter file and passes the user's choices to the Set User Parameters activity.

A12 Set User Parameters: This activity initializes the system control variables based on the user choices.

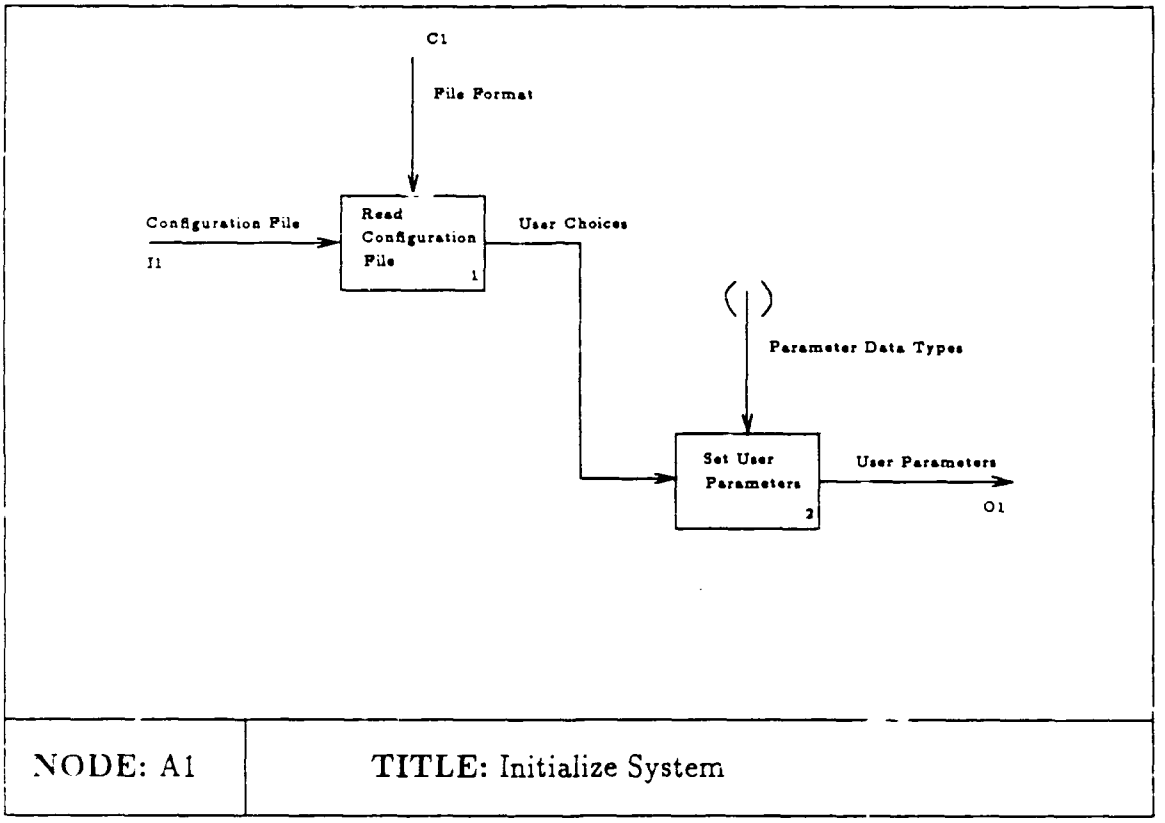

Figure 24. IDEF$_0$ A1 Diagram

A2 Process Grid

Abstract: This activity takes an arbitrarily shaped grid as input, and processes it into a rectangular format suitable for the image rendering software.

A21 Carve Grid Section: This activity cuts a portion of the input grid for processing based on the user set boundaries.

A22 Build Rectangular Grid: This activity constructs the rectangular grid onto which the data will be sampled. The size, and plane spacing are determined by the user parameters.

A23 Sample Data: This activity samples the data onto the rectangular grid. The sampling method, filtering function, and filter radius multiplier are determined by the user parameters.

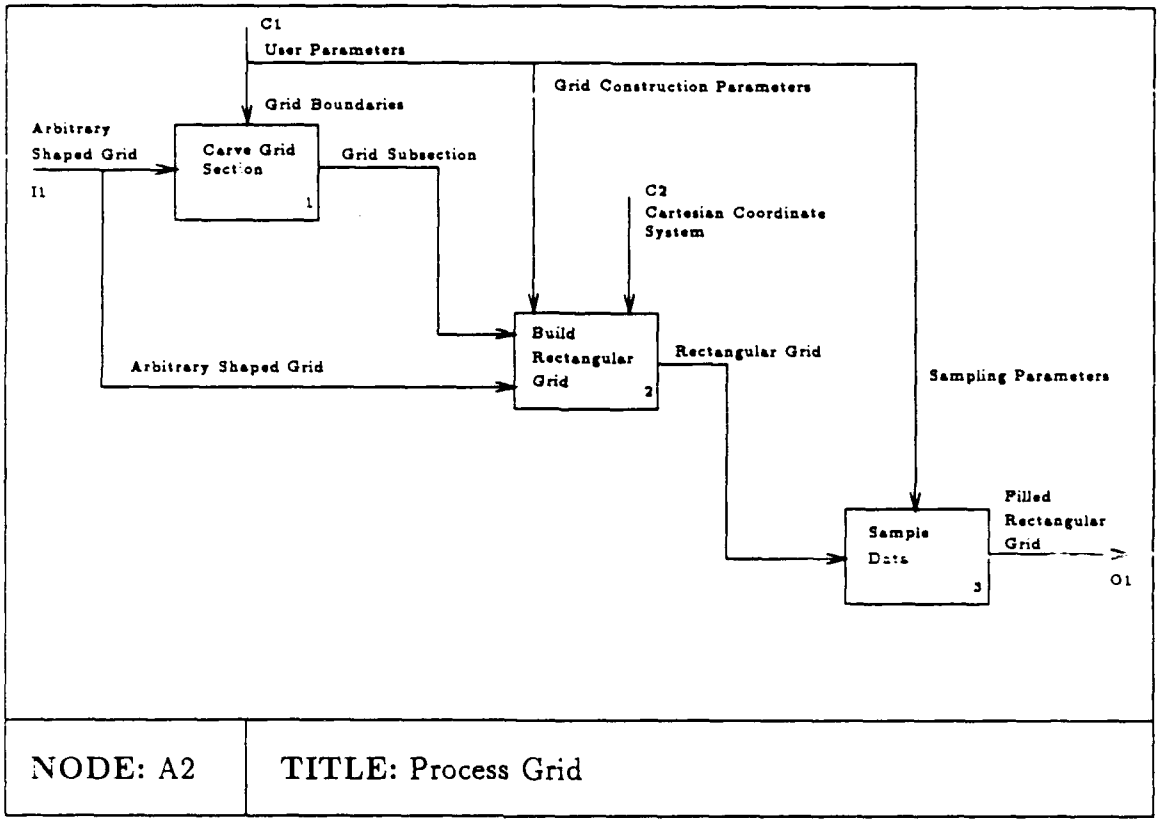

Figure 25. IDEF$_0$ A2 Diagram

Appendix B. *System Design*

B.1 Introduction

This appendix contains the structure charts used to accomplish the detailed design for the sampling program. Each box in the charts represents a function or module in the program. The relationship between each function is illustrated by the position of the boxes with respect to each other.

The boxes at the top of each diagram perform the supervisory, or control, function for the boxes under them. This is further illustrated by the lines connecting the boxes. The arrow head on the lines point from the calling module to the called module.

Detailed Design. The charts are organized in a hierarchical fashion with each chart revealing more lower level detail of the program implementation. Figure 26 displays the highest level of the program function. The numbers in the boxes are the numbers assigned to the corresponding code modules. The numbering scheme allows the position of any module to be quickly determined. At each level of decomposition the lower level modules inherit the upper module's number, along with additional numbers to identify its position relative to every module at the same level.

As Figure 27 shows, data flow between modules is represented by open circles with a line and arrow head attached. The arrow head indicates the direction of data flow. A short description is given for each data item, but the description is not necessarily related to the structure of the data. The combination of the numbering scheme and the representation of the data flow make it very easy to trace data through the program. This feature is useful for maintaining and modifying the program.

Modules which perform input and output functions are identified by double vertical lines on either end of the box. Some modules may be called by more than

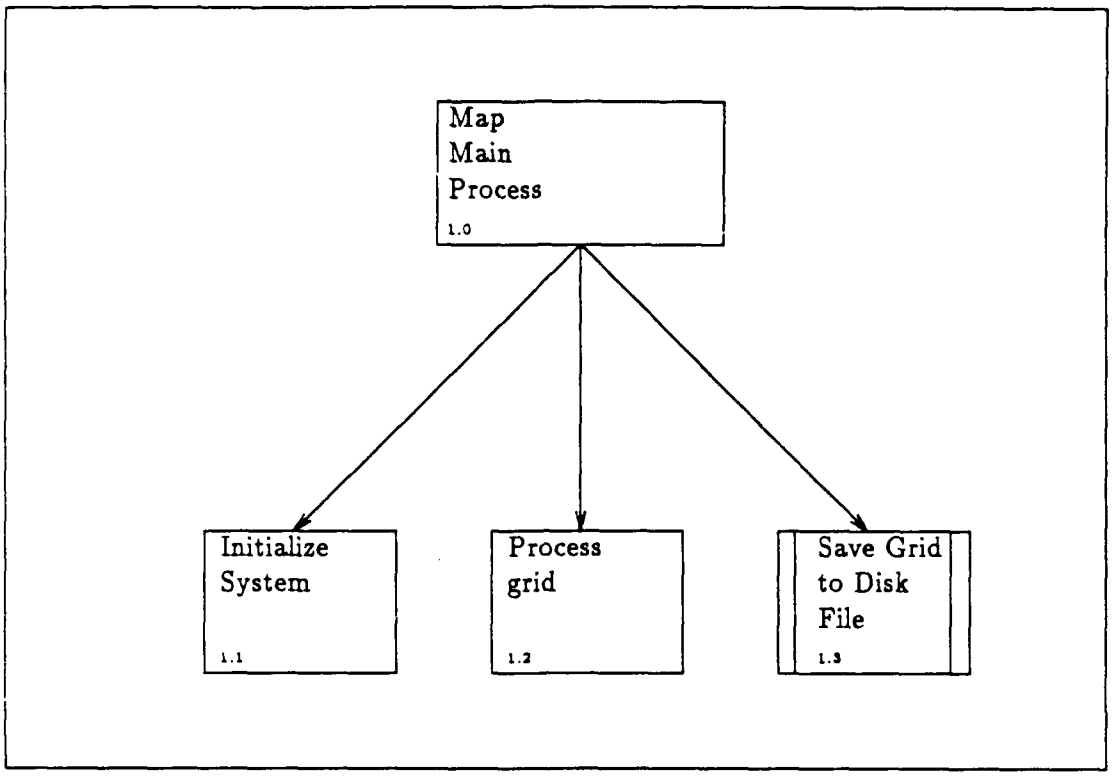

Figure 26. Level Zero Detailed Design Structure Chart

one other module. This may be shown by a diagonal line in the upper left corner of a box, or not shown at all. Module 1.2.5.1, get the three dimensional grid index in Figure 29, is called by several modules in the program. Although this is not indicated on the structure charts, is is clearly documented in the source file.

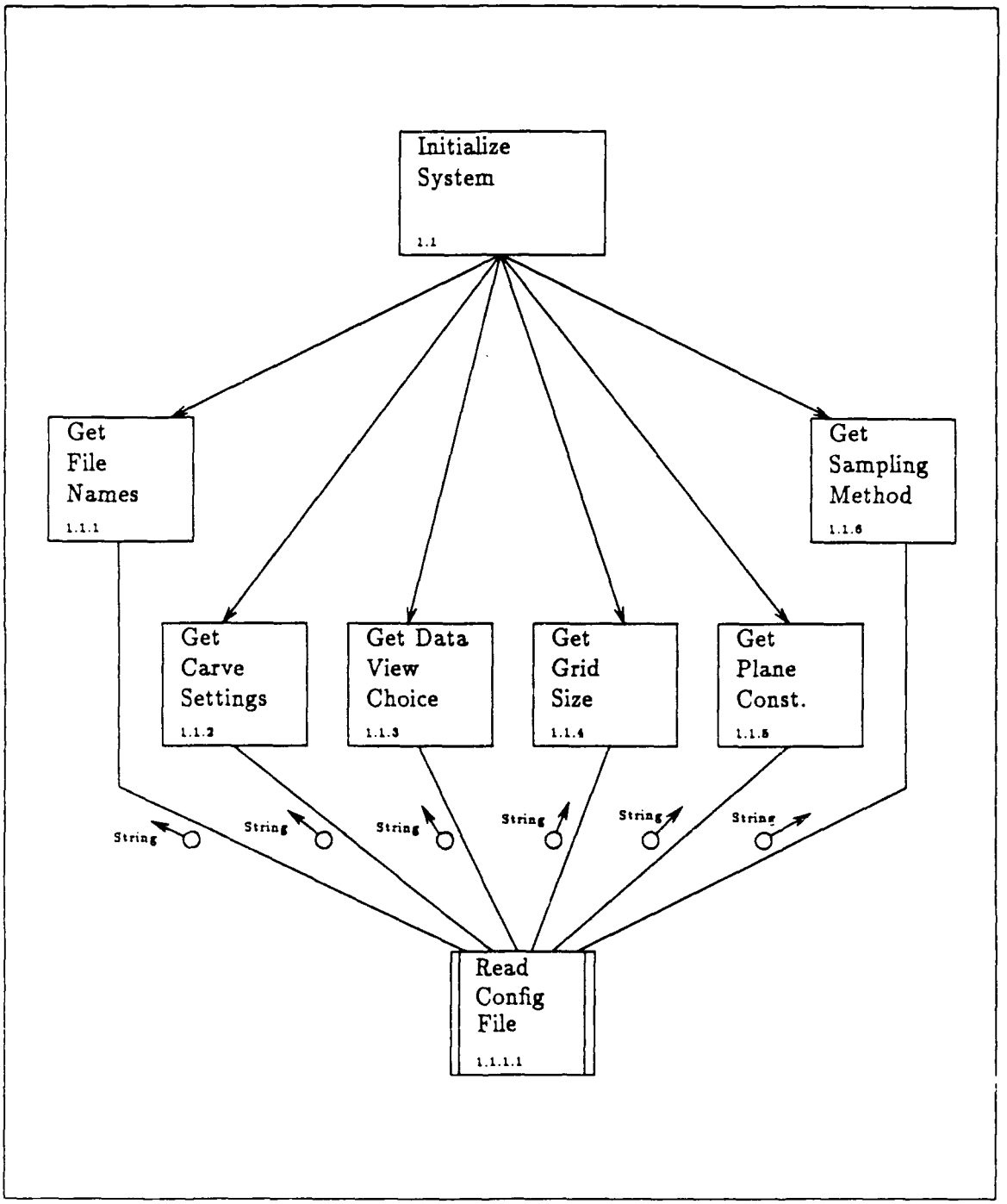

Figure 27. The Initialization Detailed Design

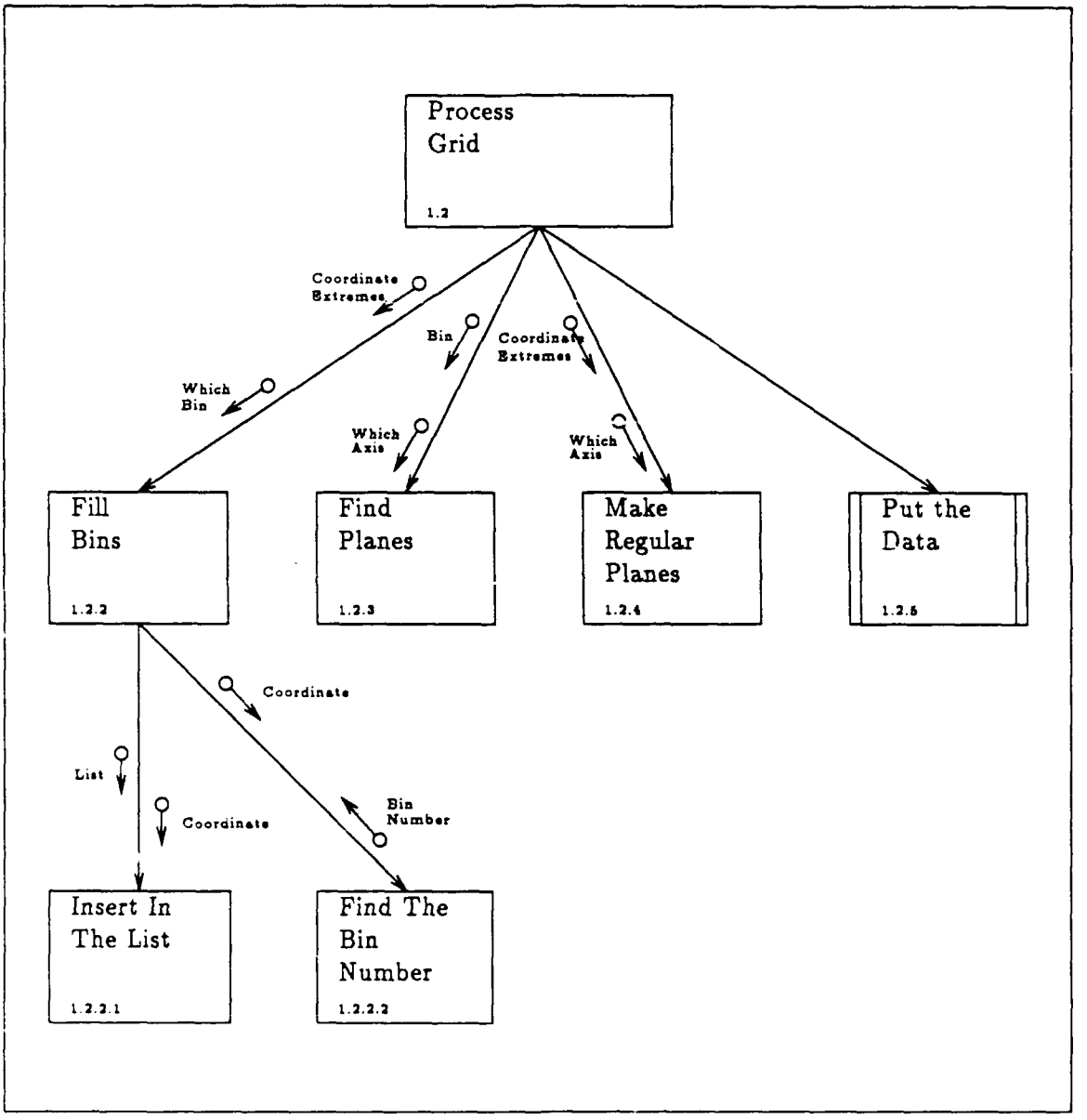

Figure 28. The Grid Processing Detailed Design

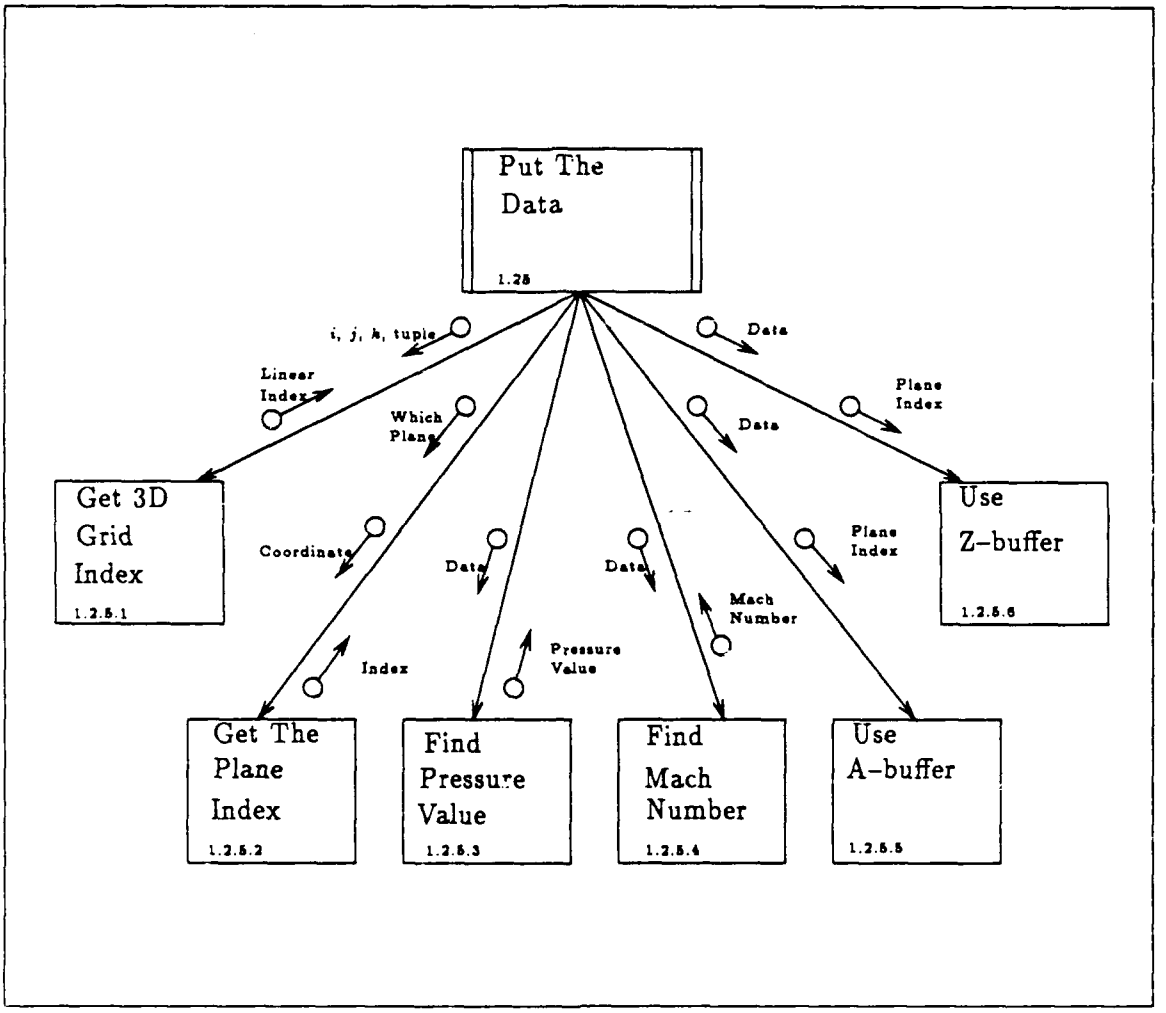

Figure 29. The Data Placement Detailed Design

Appendix C. *Unix Manual Page*

A Unix manual page was written for the mapping program, and a copy of that page is contained in this appendix. The manual page conforms to the format required by Unix, and can be installed on any Unix system.

The manual page is meant to provide a quick, on line reference for the mapping program. The file name for the manual page is *map.1*, and is included as part of the source code package. Suggestions for improvement, or change, are welcome, and should be sent to the Electrical and Computer Engineering Department, School of Engineering, Air Force Institue of Technology at Wright-Patterson Air Force Base in Dayton Ohio.

NAME

map - create a rectangular grid from scattered data

SYNOPSIS

map [-f] configuration filename

DESCRIPTION

The map program creates a rectangular grid, and samples input data from the files specified in configuration file. The program will build a grid of variable, or constant plane spacing, and create an output file conforming to VIPER's input requirements.

Two sampling methods are offered; A-Buffer, and Z-Buffer. If A-Buffer is chosen, either a density weighting, or a Gaussian filter can be specified.

The program expects the grid input file structure in the form used by the the Computational Fluid Dynamics Group at WPAFB. These grids come in two files: one for the grid indices, and one for the grid data. Before the map program can process these files, the grid data file should be transformed using trsform (part of the tools package).

The -f option is the only command line option available with the map program. It forces the program to maintain the input data coordinate ranges when using the variable spaced method for plane construction.

The user configurable parameters for the program should appear in the configuration file. The parameters are listed below in the order expected by the map program. Each parameter is briefly explained, and a sample configuration file is given in the EXAMPLES section.

USER CONFIGURABLE PARAMETERS

Number	Explanation
1	The input grid index file (paths acceptable)
2	The input grid data file (paths acceptable)
3	The output rectangular grid file name (paths acceptable)
4	A '1' or '0' for carving the input grid. A '1' means carve the grid, a '0' means do not carve the grid. If carving the grid, the next six lines of the configuration file should have the i, j, and k lower and upper bounds, (one bound per line, in the given order).

5 A '1' or '0' for the viewing parameter. A '1' means view pressure numbers, a '0' means view mach numbers.

6 A '1' or '0' for the number of planes to use in constructing the grid. A '1' means use the default (100 per axis), a '0' means other values are specified. If other values are used, all three should appear on the next line (x, y, and z).

7 A '1' or '0' for plane spacing method. A '1' means use constant spacing, a '0' means use variable spacing.

8 A '1' or '0' for the sampling method. A '1' means use Z-buffer, a '0' means use A-buffer.

9 A floating point value for the filter radius multiplier.

10 If a-buffer sampling was selected, a '1' or '0' to specify the filtering function. A '1' means use the Gaussian filter, a '0' means use the density weighting filter.

11 If the Gaussian filter is used, a floating point number for the number of standard deviations spanning the filter radius.

EXAMPLES

```
map -f test.cfg
```

SAMPLE CONFIGURATION FILE

Parameter	File Comments
grd109.bin	* The input grid index file
mq109.bin	* The input grid data file
mach1.grd	* The grid output filename
1	* Indicates carve the grid
0	* i lower bound
79	* i upper bound
0	* j lower bound
60	* j upper bound
20	* k lower bound

```
65              * k upper bound
0               * Produce a mach shell grid
1               * Use the default number of planes
1               * Use constant plane spacing method
0               * Use the A-buffer sampling technique
1.0             * The filter radius multiplier
1               * Use the Gaussian sampling filter function
6.0             * The filter radius is 6.0 standard devia-
                  tions long
```

Bibliography

1. Bridges, David J. *Volumetric Rendering Techniques for the Display of Three-Dimensional Aerodynamic Flow Field Data*. MS thesis, AFIT/GCS/ENG/88D-2. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, 1988.
2. Carpenter, Loren. "The A-buffer, an Antialiased Hidden Surface Method," *Computer Graphics*, *18*(3):103–108 (July 1984).
3. Cendes, Zoltan J. "Unlocking the Magic of Maxwell's Equations," *IEEE Spectrum*, *26*(4):29–33 (April 1989).
4. Cook, Robert L., et al. "Distributed Ray Tracing," *Computer Graphics*, *18*(3):137–145 (July 1984).
5. Drebin, Robert A., et al. "Volume Rendering," *Computer Graphics*, *22*(4):65–74 (August 1988).
6. Foley, J. D. and A. Van Dam. *Fundamentals of Interactive Computer Graphics*. Reading, Massachusetts: Addison-Wesley Publishing Company, 1984.
7. Frenkel, Karen A. "Volume Rendering," *Communications of the ACM*, *32*:426–435 (April 1989).
8. Fujimoto, Akira, et al. "ARTS: Accelerated Ray-Tracing," *IEEE Computer Graphics and Applications*, *6*(4):16–25 (April 1986).
9. Glassner, Andrew. "Introduction to Ray Tracing." Course notes, Siggraph 1987.
10. Hartrum, Thomas C. "System Development Documentation Guidelines and Standards." Technical Report, Air Force Institute of Technology, January 1989. Draft #4.
11. Johnson, E. Ruth and Charles E. Mosher Jr. "Integration of Volume Rendering and Geometric Graphics." In *ACM SIGGRAPH '89 Course #28, State of the Art in Data Visulization*, pages VIII–14 - VIII–19, July 1989.
12. Kroos, Kenneth A. "Computer Graphics Techniques for Three–Dimensional Flow Visualization," *Frontiers in Computer Graphics* (1985).
13. Levoy, Marc. "Display of Surfaces from Volume Data," *IEEE Computer Graphics and Applications*, *8*(3):29–37 (November 1988).
14. Lorenson, William E. and Harvey E. Cline. "Marching Cubes: A High Resolution 3D Surface Construction Algorithm," *Computer Graphics*, *21*(4):163–169 (July 1987).
15. McCormick, Bruce M., et al. "Visualization in Scientific Computing," *Computer Graphics*, *21*(6):1–14 (November 1987).

16. Newman, William M. and Robert F. Sproull. *Principles of Interactive Computer Graphics*. New York: McGraw-Hill Book Company, 1979.

17. Pierce, J. R. and J. E. Karlin. "Reading Rates and the Information Rate of a Human Channel," *Bell System Technical Journal*, *36*:497–516 (March 1957).

18. Pressman, Roger S. *Software Engineering, A Practitioner's Approach*. New York: McGraw–Hill Book Company, 1987.

19. Scherr, Stephen. Mathemetician. Personal Interview. WRDC/FIMM, Computational Aerodynamics Group, Aeromechanics Division, Flight Dynamics Lab, Wright Research and Development Center, WPAFB OH, 18 September 1989.

20. Smith, R., et al. *Visualization of Computer-Generated Flow Fields: Flow Visualization, Volume 3*. Washington, D. C.: Hemisphere Publishing Company, 1985.

21. Thompson, Joe F., et al. *Numerical Grid Generation*. New York: North-Holland, 1985.

22. Upson, Craig, et al. "The Application Visualization System: A Computational Environment for Scientific Visualization," *IEEE Computer Graphics and Applications*, *9*(4):30–41 (July 1989).

23. Upson, Craig and Michael Keeler. "V-Buffer: Visible Volume Rendering," *Computer Graphics*, *22*(4):59–64 (August 1988).

24. Upson, Craig and David Kerlick. "Volumetric Visualization Techniques." In *ACM SIGGRAPH '89 Course #13, 2D and 3D Visual Workshop*, pages 1–86, July 1989.

25. Watson, Val, et al. "Use of Computer Graphics for Visualization of Flow Fields." In *ACM SIGGRAPH '89 Course #28, State of the Art in Data Visualization*, pages IV–13 – IV–18, July 1989.

26. Webster, Dr. Phil. Aerodynamic Engineer. Personal Interview. WRDC/FIMM, Computational Aerodynamics Group, Aeromechanics Division, Flight Dynamics Lab, Wright Research and Development Center, WPAFB OH, 20 October 1989.

27. Westover, Lee. "Interactive Volume Rendering." Department of Computer Science, The University of North Carolina at Chapel Hill, Chapel Hill, North Carolina.

28. Whitted, Turner. "An Improved Illumination Model for Shaded Display," *Communications of the ACM*, *23*(6):343–349 (June 1980).

29. Winkelmann, A. E. and C. P. Tsao. *Flow Visualization Using Computer-Generated Color Video Displays of Flow Field Survey Data: Flow Visualization, Volume 3*. Washington, D. C.: Hemisphere Publishing Company, 1985.

Vita

First Lieutenant Raymond P. Lentz, III [redacted] He graduated from Menchville High School in Newport News, Virginia in 1973 and enlisted in the United States Air Force. He served as an aircraft mechanic from 1973 until 1979 when he was honorably discharged.

Lieutenant Lentz returned to the Air Force in 1981 to pursue a selection for the Airman Education and Commissioning program. In 1983 he entered the University of South Carolina. He graduated with honors in 1985 and received a Bachelor of Science Degree in Electrical and Computer Engineering.

Upon commissioning as a second lieutenant on April 3, 1986 Lieutenant Lentz was assigned to the 6585^{th} Test Group at Holloman Air Force Base in New Mexico. There he served as an Advanced Plans Engineer, and Radar Cross Section Analyst until May, 1988.

In June, 1988 Lieutenant Lentz entered the School of Engineering, Air Force Institute of Technology to pursue a Master of Science Degree in Computer Engineering. He is married to Sombhong (Nedpirom) Lentz, of Bangkok, Thailand.

[redacted]

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GCE/ENG/89D-4	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION School of Engineering 6b. OFFICE SYMBOL (If applicable) AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Air Force Institute of Technology Wright-Patterson AFB OH 45433-6583	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION AFWL 8b. OFFICE SYMBOL (If applicable) SCP	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code) Kirtland AFB NM 87117	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO / WORK UNIT ACCESSION NO

11. TITLE (Include Security Classification)

SAMPLING SCATTERED DATA ONTO RECTANGULAR GRIDS FOR VOLUME VISUALIZATION (UNCLASSIFIED)

12. PERSONAL AUTHOR(S)
Raymond P. Lentz, III, B.S., 1st Lt, USAF

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
MS Thesis	FROM _____ TO _____	1989 December	98

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Computer Graphics
12	05		Flow Fields
07	04		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Thesis Advisor: Phil Amburn, Maj, USAF
Professor of Computer Systems

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL Phil Amburn, Maj, USAF	22b. TELEPHONE (Include Area Code) (513) 255-2040 22c. OFFICE SYMBOL AFIT/ENG

DD Form 1473, JUN 86 *Previous editions are obsolete.*

Three dimensional arrays of scalar data representing spatial volumes arise in many scientific applications. Analysis of this type of data is difficult because of the size of the data sets. Computer graphics techniques for rendering images of three dimensional data have been developed recently.

In computational fluid flow analysis, methods for constructing three dimensional numerical grids are being refined. This technique is particularly suited for simulations involving finite element analysis. The three dimensional grids produced by these methods are generally not rectangular in shape.

Many graphics methods for rendering three dimensional volume images take advantage of the physical structure of rectangular grids. Because finite element analysis is useful in fields other than fluid flow analysis and the numerical grid has promising applications, methods for handling arbitrarily shaped data grids are needed.

This thesis investigation develops a method for sampling data in virtually any form onto rectangular grids. Two sampling methods are developed and implemented using two different sampling filters. The results were successful in rendering images of both fabricated data and data from a fluid flow simulation.

Input data distribution characteristics affecting the sampling techniques were identified, and possible solutions are provided. The goal of this study was to determine if data could be successfully sampled from a grid of arbitrary shape onto a grid of rectangular shape. The results indicate that this is indeed possible.